

Scalability Evaluation of Data Transfer Framework for Multi-Component Applications

JIE YIN^{1,a)} BALAZS GEROFI^{1,b)} ATSUSHI HORI^{1,c)} YUTAKA ISHIKAWA^{1,d)}

Abstract:

Multi-component workflows play a significant role in High-Performance Computing and Big Data applications. They usually contain multiple, independently developed components that execute side-by-side to perform sophisticated computation and exchange data through file I/O over the parallel file system. However, file I/O can become an impediment in such systems and cause undesirable performance degradation due to its relatively low speed (compared to the interconnect fabric), which is unacceptable especially for applications with strict time constraints. The Data Transfer Framework (DTF) is an I/O arbitration layer working with the PnetCDF I/O library to eliminate the bottleneck by transparently redirecting file I/O operations through the parallel file system, to message passing via the high-speed interconnect fabrics between coupled components. Scalable and high-speed data transfer between components can be thus easily achieved with minimal development effort by using DTF. However, previous work provides insufficient scalability evaluation of DTF. In order to comprehensively evaluate the scalability of an I/O middleware like DTF and highlight its major advantages, we have designed an ensemble-based I/O benchmark that adopts the I/O model of the real-time weather forecasting application called SCALE-LETKF and present the scalability evaluation results of DTF against file I/O on two supercomputers, Fugaku and Oakforest-PACS, respectively. We provide insights into DTF's scalability and performance enhancements with the intention to impact future I/O middleware design.

1. Introduction

The rapid growth of the computational speed of supercomputers creates new possibilities to solve more complex scientific problems within a reasonable time limit. It also brings opportunities for developers to pursue better performance and solutions for their High-Performance Computing (HPC) applications with different computational models. Multi-component workflow is one of the prevalent scientific computation models in High-Performance Computing, which tightly couples two independently developed components by executing side-by-side and exchanging a gigantic amount of computational data between them for the succeeding computation. Big Data Assimilation is a representative application deployed widely in many real-world systems indispensable to modern society, such as in weather forecasting systems. According to the characteristic of such workflow by binding to separately executing applications, file I/O using the parallel file system has been the traditional approach for data exchange between components. Data written by a component will be read by its couple component for the subsequent computations. I/O libraries are common approaches to generate data files and store massive datasets

in an organized and portable way. Some HPC application-oriented optimizations are usually introduced into such libraries to improve I/O performance [5], [10]. For example, Parallel netCDF (PnetCDF) [10] is one of the widely used I/O libraries which provides parallel support for accessing data stored in NetCDF format using MPI-IO [8]. However, in some multi-component applications with time constraints, massive data transfer between components through file I/O prevents the progress of achieving the real-timeliness due to its relatively slow processing speed. Coupling toolkits [3], [9] or transmitting data via interconnect fabrics by message passing are the two possible solutions for this bottleneck. However, several of the proposed solutions are time-consuming to be implemented into a HPC application because rewriting a large part of code is required. There exists a pressing need for an easy-to-use approach to diminish data exchange time for multi-component applications since improving data transfer should not be the problem for scientific application developers to tackle. This has been the primary motivation to design an I/O middleware such as the Data Transfer Framework (DTF) [11] for alleviating the data transfer problem and meeting the expectations stated above.

DTF is an I/O arbitration middleware aiming at reducing data exchange time between coupled components through bypassing file I/O and transmitting data via the interconnection network. It works with the Parallel netCDF (PnetCDF)

¹ RIKEN Center for Computational Science

a) jie.yin@riken.jp

b) bgerofi@riken.jp

c) ahorti@riken.jp

d) yutaka.ishikawa@riken.jp

I/O library while keeping application code intact. DTF plays a role as an arbitrator and it transparently intercepts PnetCDF calls invoked in user applications. It provides both file I/O and network based data transfer modes. All the steps required for switching from file I/O to high-speed data transfer are to write a simple configuration file, insert three intuitive DTF functions, and compile the project with the provided DTF based PnetCDF library. Section 2 gives a detailed description of DTF and its mechanism.

Previous work [11] has reported performance evaluation comparing direct data transfer using DTF against file I/O on the K Computer [7]. The K computer had its compute nodes connected by the 6D mesh/torus network Tofu [2], and equipped with a Lustre-based global parallel file system. DTF's in-depth behavior analysis has been conducted using the S3D-IO benchmark [13], and its performance for a real-world multi-component application SCALE-LETKF [12] has been disclosed. However, there are mainly three limitations addressed in the previous assessment.

- **Limited number of data sets.** SCALE-LETKF is a real-time weather forecasting application adopting an ensemble-based approach, in which the two components, SCALE and LETKF, are developed by independent research groups [12]. SCALE is in charge of ensemble forecasting while LETKF assimilates real-world observation data reported by weather radar with the output data produced by SCALE during each operation cycle. Its operation requires real-world datasets, which proved to be difficult for performing a thorough performance evaluation.
- **Inflexible problem size configuration.** The size of real-world datasets is often based on observation and its resolution, which is continuously upgraded along with the efforts of domain scientists. The fixed ensemble size in the datasets limits our ability to conduct an in-depth discussion of the I/O behavior and scalability of the I/O middleware, because the throughput of data transfer for the whole system can only be adjusted by changing the number of ensembles. It's difficult to gain a thorough insight into the scalability of the I/O middleware with respect to a specific ensemble.
- **Interleaved computation and I/O.** Real-world applications interleave computation with I/O in their code, which may hinder the accurate evaluation of I/O behavior. Computation phases cause components to enter their respective I/O phase asynchronously, which makes it difficult to identify potential I/O bottlenecks and provide further performance improvement.

To address the limitations stated above, this paper proposes an I/O benchmark that precisely follows the I/O model of SCALE-LETKF. The benchmark strips computation phases away and provides the ability to adjust the size of the overall and per-process I/O operations. We use this benchmark to evaluate both strong scaling and weak

scaling of DTF comparing to file I/O using two different supercomputer platforms, Fugaku and Oakforest-PACS [6]. We also present the speedup and efficiency comparison of DTF direct data transfer between these two supercomputers, demonstrating DTF's ability to scale over different network topologies.

The rest of this paper is organized as follows. Section 2 includes a detailed description of the mechanism behind DTF and its simple usage. Section 3 introduces the benchmarks used for the performance assessment and showcases the comparison results in detail. The current conclusion and future efforts are stated in Section 4.

2. Data Transfer Framework

Data Transfer Framework (DTF) [11] is an I/O arbitration middleware devoted to reducing the data transfer time between coupled components in a multi-component application. The principle of DTF is to silently redirect the file I/O based data exchange to message passing for the components while keeping the original application code intact and requiring the least effort to start using it. In this section, we provide a brief overview of DTF, describe how it works under the hood, and highlight its advantages.

2.1 DTF Overview

DTF works with I/O operations for files in the Network Common Data Form (NetCDF) format [15], which is designed for storing and organizing massive data in a portable and efficient way. As shown in Figure 1, DTF is integrated with the parallelized implementation of the NetCDF format named Parallel netCDF (PnetCDF) that is built on top of MPI-IO [8], to allow multiple processes performing I/O to the same NetCDF file in parallel. PnetCDF is proven to produce an outstanding file I/O performance against other I/O libraries like serial netCDF and parallel HDF5 in terms of scalability and bandwidth [10]. According to the previous work, it only requires about 50 lines of code modification inside PnetCDF for integrating with DTF [11].

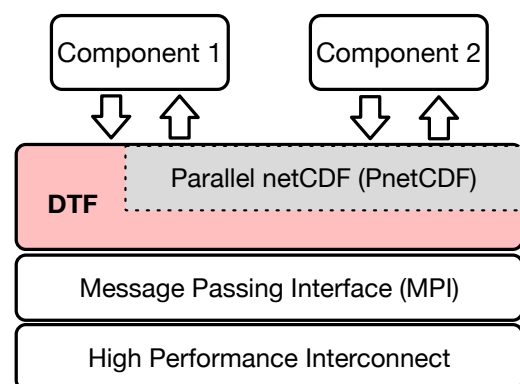


Fig. 1: An overview of the Data Transfer Framework (DTF).

During the initialization stage, DTF sets up a connection session between the two components by establishing an MPI inter-communicator. All the succeeding inter-

components communications happen through this inter-communicator. While PnetCDF interfaces being invoked during I/O sessions in each component, DTF redirects the execution flow to its embedded function calls and performs inter-component message passing instead of the original I/O processing. Its internal operations are categorized into the following two types.

2.1.1 I/O Request Handling

Several processes in the writer component will be entrusted with being *request matchers* when an I/O session starts, whose responsibilities are to collect all the posted I/O requests from other processes and perform request matching. It is configurable through an environment variable for designating the expected number of matchers. By default, the number of matchers is set into 64.

Whenever a PnetCDF function from the *get* or *put* families is invoked to access a NetCDF variable, DTF will redirect this call to its internal function and initialize an I/O request object, which stores descriptive metadata about the request like start coordinate, shape, request type and address of user buffer. Request matching happens when the user signals DTF to start transferring data between the coupled components by invoking a DTF function *dtf_transfer()*. All the processes of reader and writer will distribute their I/O request objects to the matchers of the writer, which will be further stored in matchers' read request pool and write request pool respectively. After all the request objects are received, matchers will start matching requests based on the metadata stored in the request objects. All the read requests will be processed by searching in the write request pool for the write requests that cover the requested data block. Once two requests are matched, matchers will notify the matched writer process to send the requested data blocks to its reader. DTF is well aware that two components may perform asymmetric I/O operations, which happens when the reader processes never read some data blocks that are written by the writer. Request matching perfectly solves the situation by only transferring data blocks that are explicitly requested by the reader. The writer process packs a message header with the requested data blocks and delivers this package to the matched reader by message passing. Finally, the reader process receives the package, parses the message header, and unpacks the data into its designated buffer.

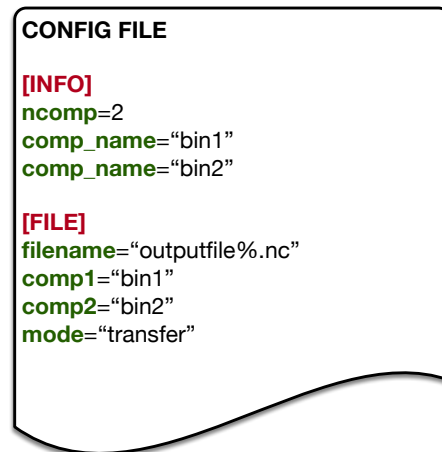
2.1.2 I/O Replaying

I/O replaying is an optimization solution inspired by the I/O pattern of some multi-component applications. These applications commonly execute multiple cycles, while each cycle performs the identical I/O operations. When I/O replaying is enabled, DTF saves the history of request matching for the first cycle, and repeat the same transfer pattern for the rest of the cycles. Request matching is skipped for the subsequent cycles and DTF can start transferring data immediately. This feature can be enabled for applications by setting the parameter *replay-io* in the configuration file.

2.2 Simple Use of DTF

DTF needs three preparation works to be done before multi-component applications can switch from file I/O to high-speed data transfer.

- **Write a configuration file.** DTF requires a simple configuration file in a given format, which is shown in Figure 2. User is expected to briefly describe the I/O dependencies between the two components and specify the mode of data transfer for each listed file pattern in key-value pairs. DTF encourages users to make the final decision on how data will be exchanged during runtime. When the name of data files do not match the listed name pattern or the file I/O mode is explicitly set, DTF will silently step aside without interfering with the I/O processing of PnetCDF. Other optional configurations are available for users to flexibly adjust DTF's behavior and fully boost data transfer between components, such as replaying I/O and buffering user data.



```

CONFIG FILE

[INFO]
ncomp=2
comp_name="bin1"
comp_name="bin2"

[FILE]
filename="outputfile%.nc"
comp1="bin1"
comp2="bin2"
mode="transfer"

```

Fig. 2: An example of a DTF configuration file.

- **Insert DTF interfaces.** Three intuitive DTF functions listed below should be inserted into the code of both components.

dtf_init(config_file, comp_name) parses user-defined configuration file *config_file*, initializes DTF and builds the inter-communicator between coupled components. This function should be called by all the processes after the MPI library is initialized.

dtf_finalize() finalizes DTF. This function should be called by all the processes before the MPI library is finalized.

dtf_transfer(file_name) starts matching collected I/O requests and transferring data between components for the file named *file_name*, therefore, it can only be called when there are I/O requests already posted. This function has to be called by all the processes that perform I/O to the data file *file_name*, which should have already be registered in the configure file *config_file*. *dtf_transfer()* is a blocking function call that returns

only when the reader receives all the data it has requested. User should decide the appropriate location to call this function in their applications.

- **Build with DTF based PnetCDF library.** Lastly, user should recompile each component with the provided PnetCDF library integrated with DTF. DTF is ready-to-use once this step is completed.

2.3 Strong Points of DTF

In this subsection, advantages of DTF are summarized into three points as follows.

Minimal changes to the original source code.

DTF keeps the original application code as intact as possible. It lifts a great weight for application developers if their multi-component applications are suffering from inefficient data exchange but can only be improved by largely modifying the I/O code.

Adjustable run-time behavior. The run-time behavior of DTF is adjustable by changing the key-value pair in the configuration file. Its flexibility keeps DTF being competent in diverse I/O scenarios.

Only explicitly requested data is transferred.

In some I/O scenarios, writer component constantly writes large amount of uncalled-for data while reader never consumes them. DTF ingeniously avoids spending time on these unwanted I/O operations by request matching.

3. An Ensemble-Based I/O Benchmark

The ensemble-based method is widely used in various geophysical applications. In an ensemble-based I/O pattern, processes of each component are organized into multiple ensembles and performing I/O to different files in each independent ensemble. In this section, we present an ensemble-based multi-component benchmark adopting the I/O pattern of a real-world weather forecasting application named SCALE-LETKF [12].

3.1 Benchmark Overview

We designed this benchmark for simulating the I/O sessions of the SCALE-LETKF. As briefly introduced in Section 1, it is a weather forecasting application with real-time requirements – that is, the execution of one cycle should finish within 30 seconds, which includes both computation time and I/O time. Reducing the I/O time as much as possible can help the application get one step closer to its timeliness target.

The I/O workflow of each iteration in each ensemble is portrayed in Figure 3. Three I/O sessions happen in each cycle between the two components with two type of files involved, which are the *history file* and *analysis file*. SCALE processes in each ensemble start every cycle by reading the

analysis file that is written by LETKF processes and subsequently write to the history file and analysis file that is read by LETKF.

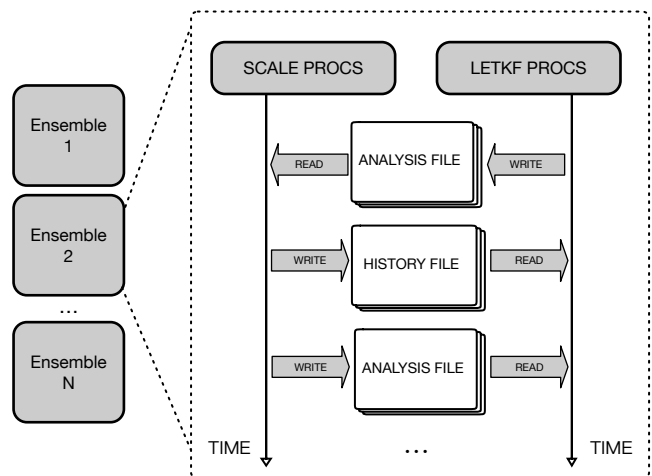


Fig. 3: An overview of SCALE-LETKF's I/O model for each cycle in each ensemble. The same ensemble-based I/O pattern is adopted in this benchmark.

Besides performing I/O in an ensemble approach, a mixture of symmetry and asymmetry between the two components is another major characteristic of this I/O model. Symmetry happens in the session while SCALE reads the analysis file and LETKF writes it. Both of them access the same number of NetCDF variables with an identical description of data access region. However, I/O behaviors of the two components become asymmetric in the sessions when SCALE writes data to the history and analysis file for LETKF. SCALE outputs data to 89 NetCDF variables of the history file and 143 variables of the analysis file, while LETKF only reads data of 20 NetCDF variables from the history file and 11 variables from the analysis file. Moreover, SCALE writes values of halo space to the analysis file while LETKF never reads them. The asymmetry incurs a large amount of execution time being wasted on the undesirable I/O operations. Fortunately, DTF perfectly diminishes this problem by request matching as introduced in Section 2.1.1.

3.2 Scalability Evaluation

This subsection introduces the scalability evaluation results of DTF compared to the PnetCDF file I/O using this benchmark. The benchmark contains multiple execution cycles, and the I/O operations performed in each cycle behave in the same pattern and data size. Therefore, we enable the *replay.io* optimization for this benchmark and measure the average data transfer time for one cycle. PnetCDF-1.7.0 library is used for all the experiments.

Supercomputer Fugaku and Oakforest-PACS (OFP) are used for these scalability evaluation experiments. Fugaku has gained the title of the world's fastest supercomputer. Its compute nodes are connected via the newly designed 6D mesh/torus interconnect called Tofu Interconnect D (ToFuD) [1]. It is equipped with a three-layered storage system

that includes a Lustre-based global file system named Fujitsu FEFS [4], [14]. Only the second layer of FEFS is used for the following experiments conducted on Fugaku. OFP is another Fujitsu built supercomputer which is managed by The University of Tsukuba and The University of Tokyo [6]. All the compute nodes are interconnected by Intel's Omni Path network and able to access a Lustre-based global parallel file system.

Due to the varied number of available nodes between the two platforms, we have designed experiments with different configurations. Currently we change the total amount of I/O by adjusting data sizes in each ensemble, while the total numbers of ensembles are fixed on both platforms. We completely follow the working model of SCALE-LETKF and assume all the processes in an ensemble are organized as a 2-D cartesian grid. Several parameters are introduced to control the shape of each grid cell, such as *IMAX* which means the length of a grid cell on the x-dimension, and *JMAX* which refers to the length on the y-dimension. The size of each grid cell also signifies the base of data volume per process. In the case of strong scaling, the overall grid size of each ensemble is configured into 1024×1024 on both platforms. In the case of weak scaling, each process performs the same amount of I/O by setting the shape of each grid cell into constants. Subsections 3.2.1 and 3.2.2 introduce experiment configurations and evaluation results in detail.

Data Transfer Time of each following plot literally means the average time interval from I/O requests posted until all the requested data delivered during one cycle in DTF direct data transfer. Because we removed all the operations irrelevant to I/O from the operating cycles, the writer and reader will post I/O requests almost at the same time. Therefore, the reported time genuinely reflects how much time the reader has to wait for the delivery of data. In the PnetCDF file I/O, *Data Transfer Time* indicates the average total time spent in writing and reading for each cycle. The Y-axis of all the figures is on a logarithmic scale for a clear and precise comparison.

3.2.1 Scalability on Fugaku

We deployed the experiments up to 5120 nodes on Fugaku with a total number of ensembles set into 10. The I/O sizes performed by each process in the strong scaling experiment are listed in Table 1. The maximum amount of I/O during each cycle performed by SCALE reaches about 189.8 GiB, and 142.3 GiB performed by LETKF. In the weak scaling experiment, the amount of I/O per process during each cycle is fixed into 77.4 MiB per SCALE process and 57.6 MiB per LETKF process.

The strong scaling and weak scaling results are presented in Figure 4. DTF considerably outperforms the PnetCDF file I/O and scales better in each case. In the strong scaling, data transfer time is reduced to 10x lower when the total number of nodes reaches 5120 compared to its starting point in the direct data transfer, while the PnetCDF file I/O can achieve 2x speedup. In the weak scaling, the DTF direct

Table 1: The amount of I/O per-process during each cycle in the strong scaling experiment on Fugaku. (MiB)

# of PROCS	SCALE		LETKF	
	WRITE AMOUNT	READ AMOUNT	WRITE AMOUNT	READ AMOUNT
320	894.4	320.5	320.5	590.2
640	449.0	160.3	160.3	295.3
1280	225.5	80.3	80.3	147.9
2560	113.7	40.3	40.3	74.2
5120	57.2	20.2	20.2	37.3

data transfer keeps a steady transfer time with a growing number of nodes, while PnetCDF file I/O performs at an increasingly slower rate.

It's necessary to mention that evaluation results on Fugaku are tentative for the current stage because it's still under development. There are a lot of spaces available for the Fugaku project to grow.

3.2.2 Scalability on Oakforest-PACS

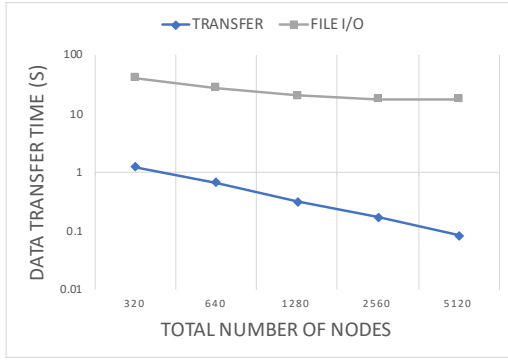
Due to the compute resource limitation, we deployed the experiments on OFP using 2048 nodes maximumly with the number of ensembles set into 8. Table 2 has listed the amount of I/O performed by each process in the strong scaling evaluation. The total amount of I/O during each cycle performed by SCALE and LETKF are approximately 151.4 GiB and 113.7 GiB respectively. In the weak scaling experiment, the amount of I/O per process during each cycle is configured into 153.6 MiB per SCALE process and 113.7 MiB per LETKF process.

Table 2: The amount of I/O per-process during each cycle in the strong scaling experiment on OFP. (MiB)

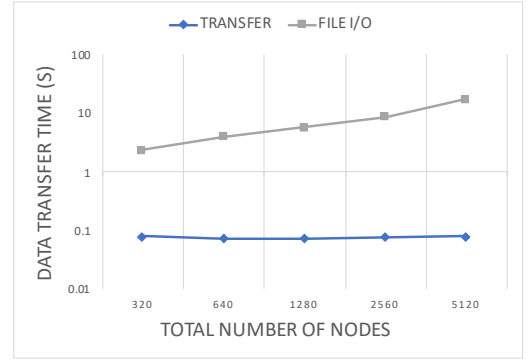
# of PROCS	SCALE		LETKF	
	WRITE AMOUNT	READ AMOUNT	WRITE AMOUNT	READ AMOUNT
64	3563.0	1281.0	1281.0	2359.0
128	1785.1	640.5	640.5	1179.5
256	894.4	320.2	320.2	589.7
512	449.0	160.1	160.1	294.8
1024	225.5	80.1	80.1	147.4
2048	113.7	40.0	40.0	73.7

Figure 5 exhibits the scalability difference between DTF and PnetCDF file I/O using OFP. DTF significantly outperforms the PnetCDF file I/O and scales better as expected. DTF delivers data in a notably faster rate in strong scaling, while keeps a stable data transfer time in weak scaling. The PnetCDF file I/O scales distinctly worse in both scaling experiments.

For clarification purposes, we present the speedup comparison of DTF direct data transfer between Fugaku and OFP as shown in Figure 6. We carried out the experiment using the same configuration as introduced in Subsection 3.2.2 on both platforms. According to the evaluation results, DTF is constantly scalable on both supercomputers in the experiments of strong scaling and weak scaling.

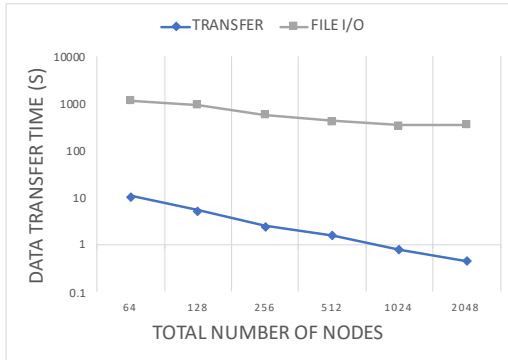


(a) Strong scaling

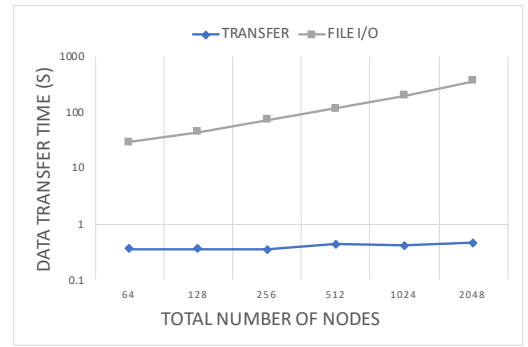


(b) Weak scaling

Fig. 4: Scalability of DTF compared to the PnetCDF file I/O on Fugaku.

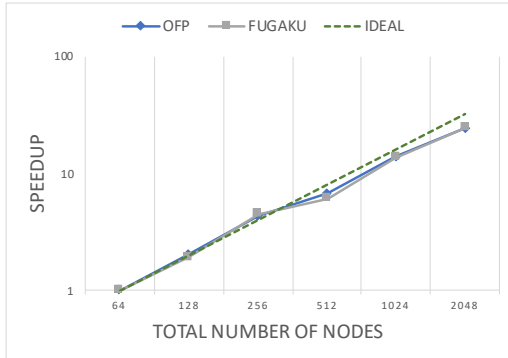


(a) Strong scaling

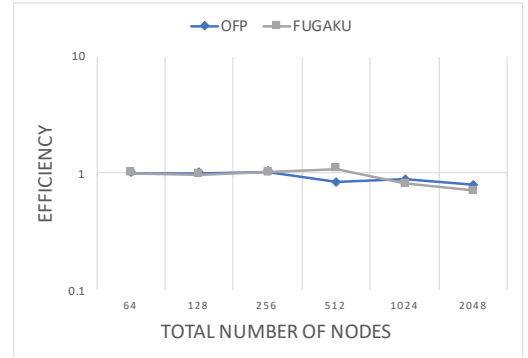


(b) Weak scaling

Fig. 5: Scalability of DTF compared to the PnetCDF file I/O on Oakforest-PACS.



(a) Strong scaling speedup



(b) Weak scaling efficiency

Fig. 6: Speedup and efficiency comparison of direct data transfer between OFP and Fugaku

DTF gains almost linear speedup in the strong scaling experiments and performs in excellent weak scaling efficiency on both platforms.

4. Conclusion

This work presents an ensemble-based multi-component I/O benchmark and comprehensive results of scalability evaluations for the direct data transfer using Data Transfer Framework (DTF) compared against PnetCDF file I/O. According to the evaluation results collected on two supercomputers, Fugaku and Oakforest-PACS (OFP), DTF is found to have significant scalability advantage over the PnetCDF

file I/O both for strong scaling and weak scaling. Multi-component applications can easily benefit from DTF due to its minimal code change requirements. Additionally, comparison results of strong scaling speedup and weak scaling efficiency between Fugaku and OFP have been presented. They both have superior scalability under the direct transfer mode of DTF. The results presented in this work could be informative for future I/O middleware design.

Our future effort will focus on characterizing the scalability and performance of DTF using applications that have irregular I/O behaviors, to which replaying I/O is inapplicable. We believe that request matching has space to be

optimized when the total number of I/O requests becomes large, and load balancing between matcher processes should be revisited. With the proposed benchmark in Section 3, we are able to conduct in-depth investigation of the behavior and scalability of DTF alike I/O middleware and propose further optimization to the current approach.

5. Acknowledgement

This work has been partially funded by JST AIP Grant Number JPMJCR19U2 and MEXT's program for the Development and Improvement of Next Generation Ultra High-Speed Computer Systems under its subsidies for operating the Specific Advanced Large Research Facilities in Japan.

References

- [1] Ajima, Y., Kawashima, T., Okamoto, T., Shida, N., Hirai, K., Shimizu, T., Hiramoto, S., Ikeda, Y., Yoshikawa, T., Uchida, K. et al.: The tofu interconnect D, *2018 IEEE International Conference on Cluster Computing (CLUSTER)*, IEEE, pp. 646–654 (2018).
- [2] Ajima, Y., Sumimoto, S. and Shimizu, T.: Tofu: A 6D mesh/torus interconnect for exascale computers, *Computer*, No. 11, pp. 36–40 (2009).
- [3] Craig, A. P., Jacob, R., Kauffman, B., Bettge, T., Larson, J., Ong, E., Ding, C. and He, Y.: CPL6: The new extensible, high performance parallel coupler for the Community Climate System Model, *The International Journal of High Performance Computing Applications*, Vol. 19, No. 3, pp. 309–327 (2005).
- [4] Fujitsu: FEFS: Scalable Cluster File System, <https://www.fujitsu.com/downloads/TC/sc11/fefs-sc11.pdf>.
- [5] Howison, M.: Tuning hdf5 for lustre file systems (2010).
- [6] Joint Center for Advanced HPC (JCAHPC): Basic Specification of Oakforest-PACS, http://jcahpc.jp/eng/ofp_intro.html.
- [7] Kurokawa, M.: The K computer: 10 Peta-FLOPS supercomputer, *The 10th International Conference on Optical Internet (COIN2012)*, IEEE (2012).
- [8] Lang, S., Carns, P., Latham, R., Ross, R., Harms, K. and Allcock, W.: I/O performance challenges at leadership scale, *Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis*, IEEE, pp. 1–12 (2009).
- [9] Larson, J., Jacob, R. and Ong, E.: The model coupling toolkit: a new Fortran90 toolkit for building multiphysics parallel coupled models, *The International Journal of High Performance Computing Applications*, Vol. 19, No. 3, pp. 277–292 (2005).
- [10] Li, J., Liao, W. k., Choudhary, A., Ross, R., Thakur, R., Gropp, W., Latham, R., Siegel, A., Gallagher, B. and Zingale, M.: Parallel netCDF: A high-performance scientific I/O interface, *SC'03: Proceedings of the 2003 ACM/IEEE conference on Supercomputing*, IEEE (2003).
- [11] Martsinkevich, T. V., Gerofi, B., Lien, G. Y., Nishizawa, S., Liao, W. k., Miyoshi, T., Tomita, H., Ishikawa, Y. and Choudhary, A.: DTF: An I/O Arbitration Framework for Multi-component Data Processing Workflows, *International Conference on High Performance Computing*, Springer, pp. 63–80 (2018).
- [12] Miyoshi, T., Lien, G. Y., Satoh, S., Ushio, T., Bessho, K., Tomita, H., Nishizawa, S., Yoshida, R., Adachi, S. A., Liao, J. et al.: "Big data assimilation" toward post-petascale severe weather prediction: An overview and progress, *Proceedings of the IEEE*, Vol. 104, No. 11, pp. 2155–2179 (2016).
- [13] Northwestern University: The S3D-IO Benchmark, <https://github.com/wkliao/S3D-IO>.
- [14] RIKEN Center for Computational Science: Fugaku Supercomputer, <https://www.r-ccs.riken.jp/en/fugaku/project/outline>.
- [15] Unidata: Network Common Data Form (NetCDF), <https://www.unidata.ucar.edu/software/netcdf>.