Received: 29 November 2016 | Revised: 26 March 2017 | Accepted: 27 March 2017

DOI: 10.1002/cpe.4161

WILEY

SPECIAL ISSUE PAPER

A flexible I/O arbitration framework for netCDF-based big data processing workflows on high-end supercomputers

Jianwei Liao[1,2] | Balazs Gerofi[3] | Guo-Yuan Lien[3] | Takemasa Miyoshi[3] | Seiya Nishizawa[3] | Hirofumi Tomita[3] | Wei-Keng Liao[4] | Alok Choudhary[4] | Yutaka Ishikawa[3]

[1]College of Computer and Information Science, Southwest University of China, Chongqing, China
[2]State Key Laboratory for Novel Software Technology, Nanjing University, Jiangsu, China
[3]RIKEN Advanced Institute for Computational Science, Kobe, Japan
[4]Department of Electrical Engineering and Computer Science, Northwestern University, Evanston, IL, USA

Correspondence
Jianwei Liao, College of Computer and Information Science, Southwest University of China, Tianshen Road No. 2, Beibei, Chongqing, China.
Email: liaojianwei@il.is.s.u-tokyo.ac.jp

Funding information
MEXTs program for the Development; Improvement of Next Generation Ultra High-Speed Computer Systems; National Natural Science Foundation of China, Grant/Award Number: 61303038; Opening Project of State Key Laboratory for Novel Software Technology, Grant/Award Number: KFKT2016B05; RIKEN Advanced Institute for Computational Science through the HPCI System Research project, Grant/Award Number: hp150019

Summary

On the verge of the convergence between high-performance computing and Big Data processing, it has become increasingly prevalent to deploy large-scale data analytics workloads on high-end supercomputers. Such applications often come in the form of complex workflows with various different components, assimilating data from scientific simulations as well as from measurements streamed from sensor networks, such as radars and satellites. For example, as part of the Flagship 2020 (post-K) supercomputer project of Japan, RIKEN is investigating the feasibility of a highly accurate weather forecasting system that would provide a real-time outlook for severe guerrilla rainstorms. One of the main performance bottlenecks of this application is the lack of efficient communication among workflow components, which currently takes place over the parallel file system.In this paper, we present an initial study of a direct communication framework designed for complex workflows that eliminates unnecessary file I/O among components. Specifically, we propose an I/O arbitration layer that provides direct parallel data transfer (both synchronous and asynchronous) among job components that rely on the netCDF interface for performing I/O operations. Our solution requires only minimal modifications to application code. Moreover, we propose a configuration file–based approach that allows users to specify the desired data transfer pattern among workflow components, offering a general solution for different application contexts. We present a preliminary evaluation of the proposed framework on the K Computer (running on up to 4800 compute nodes) using RIKEN's experimental weather forecasting workflow as a case study.

KEYWORDS
asynchronous transfer, big data processing, customizability, netCDF, parallel direct data transfer, real time

1 | INTRODUCTION

With the accelerating convergence between high-performance computing (HPC) and a new generation of Big Data technologies, high-end supercomputers are increasingly being leveraged for processing the unprecedented amount of data scientific simulations and sensor networks produce.[1] Consequently, the HPC community has been heavily focusing on how to provide the appropriate execution environment for Big Data processing workloads on large scale HPC systems.

A motivating example, as well as our case study in this paper, is SCALE-LETKF,[2–4] a complex weather forecasting application that is being developed at RIKEN. With the next generation Japanese flagship supercomputer (post-K) as its primary target platform, SCALE-LETKF is intended to provide high-resolution, real-time weather forecasting of severe guerrilla rainstorms* in Japan. Similar to other operational weather forecasting applications, the SCALE-LETKF mainly consist of 2 components developed separately: a numerical weather prediction (NWP) model and a data assimilation system. The NWP model used here is the SCALE-LES (scalable computing for advanced library and environment–LES[5,6]), which simulate the time evolution of the weather-related atmosphere and land/sea surfaces based on physical equations (hereafter, Simulation). Meanwhile, the data assimilation method used here is the local ensemble transform

*The term of guerrilla rainstorms is a Japanese expression used to describe a short, localized, sudden downpour of over 100 mm of rain per hour.

Kalman filter (LETKF[7]), which assimilate observation data taken from the real world into the simulated state to produce a better initial condition for the model (hereafter, Assimilation). The 2 components run in a cyclic way: after the simulation finishes, the data assimilation starts taking the results from the simulation as its input data, and after the data assimilation finishes, the simulation of the next cycle follows, depending on the results from the data assimilation. Additionally, since the observation data are required in each cycle, they need to be streamed directly into RIKEN's supercomputing facility when executing the workflow in real time.

Both the simulation (SCALE) and data assimilation (LETKF) components in the current workflow rely on netCDF for I/O operations using the parallel file system. netCDF is a self-describing, portable, scalable, appendable, and shareable file format, which is widely used to exchange array-oriented scientific data, such as grids and time-series.[8] Historically, the decision for file-based data exchange was mainly driven by the fact that these models are being developed and maintained by independent research entities and it has been strongly desired not to modify either of the component models purely for the purpose of building a coupled forecasting system.

The prediction of guerrilla heavy rains, however, is a strictly time-constrained procedure, and we identified that file I/O-based data transfer between the 2 components is one of the hindering factors for acquiring the needed realtimeness. A large number of coupling tools, targeting effective integration of separately developed models or applications, have been proposed[9–15]; nevertheless, all of them require numerous modifications to the applications.

Our main focus in this paper is to provide an I/O arbitration framework that can enable high-performance, direct data exchange among workflow components, which process large amounts of data and use netCDF for their underlying data format. Furthermore, we seek to provide a solution that retains the original netCDF API and requires only minimal changes to existing application code. On the whole, this paper makes the following contributions:

- General I/O arbitration middleware. We propose a general I/O arbitration middleware, ie, a software library that enables direct parallel data transfer among workflow components that use netCDF for their data representation. As a result, our middleware benefits the integration of existing, separately developed models for solving complicated problems. Individual models or applications are usually developed to tackle specific scientific issues, and easy integration of existing models into complex workflows enables solving more intricate problems. The proposed I/O middle software offers a general communication mechanism for intermodel (or interapplication) communication, which is beneficial for integrating existing models and applications to overcome new problems, particularly with focus on Big Data processing.
- Accelerated data exchange in coupled systems. Compared to file-based data exchanging over the parallel file system, the proposed middleware adopts communication pattern-based optimization to efficiently support direct data transfer. Therefore, it shortens the time required for exchanging data among the components of coupled systems so that rigid time constraints of real-time applications can be satisfied.
- Asynchronous data transfer. To overlap data transmission with computation, we propose an asynchronous data transfer scheme in the middleware. Specifically, when internal memory buffer becomes full at the sender process, part of output data is transferred to the destination processes immediately. Data transfer is performed by a background thread and can proceed in parallel with the subsequent computation phase of the application. Consequently, in addition to enabling asynchronous read operations on the receiver side, the time required for data transfer after computation on the sender side can be also significantly reduced. For a detailed description of the asynchronous data transfer extension, refer to Section 3.4.
- Customizable data distribution. We propose a mechanism that enables users to specify the data transfer pattern among workflow components. Users are required to provide a configuration file with a high level description of the desired data transfer pattern among different components of the application. The middleware interprets the description at initialization time and performs data transfer according to the specification. Various data transfer patterns (eg, one-to-one, one-to-all, and all-to-all) are supported. For more information on the configuration file, see Section 3.6.

Note that, as an extension to our previous work,[16] only the last 2 of contributions are new in this paper. We perform a preliminary evaluation of the direct data transfer mechanism using the SCALE-LETKF weather forecasting workflow, a representative example of Big Data assimilation applications. As our measurements indicate, the proposed framework can provide an order of magnitude faster I/O performance compared to file system–based data exchange, while it remains highly transparent at the application source code level. Moreover, asynchronous data transfer further reduces the time required for data transmission, in contrast to the previously proposed, synchronized scheme.

The remainder of the paper is structured as follows. Related work is described in Section 2. The design and implementation of the proposed middleware are explained in Section 3. Section 4 introduces the evaluation methodology and discusses experimental results. At last, concluding remarks are given in Section 5.

2 | RELATED WORK

In weather forecasting and geoscientific systems, individual models usually have their limited focus and are in charge of analyzing a specific phenomenon. On the other hand, a practical forecasting system takes various aspects into account, and thus, it normally uses several existing models to achieve its final goal rather than developing everything from scratch. This section introduces related work focusing on coupling existing models or applications, as well as on related work about conducting data transfer among the component models or applications in such systems.

Integration mechanisms for individual models. The intricate global climate problems motivate researchers from different scientific disciplines to integrate existing multiphysics computation models or applications for exhaustive modeling by using a software framework or a coupled system.[17] The Model Coupling Toolkit[13] is a library providing

routines and datatypes for creating a coupled system, and it is mainly used in Community Climate System Model.[14] Hereafter, S. Valcke et al[18] have summarized major coupling technologies used in Earth system modeling, and their paper shows common features of the existing coupling approaches including the functionality to communicate and regrid data.

The OASIS coupler is another related study (the latest version is OASIS3), which is able to process synchronized exchanges of coupling information generated by different components in a climate system, and the coupler mediates communication among the components.[19] But the OASIS coupler has its own interface and is not a solution for general cases. Armstrong et al[20] have designed an approach to separate the code of models from the coupling infrastructure, but it does not provide coupling functions such as data transfer. However, it enables users to choose the underlying coupling functions from other couplers, such as the OASIS coupler. Besides, there are numerous existing middlewares for coupling specific models, such as ESMF,[21] the FMS coupler,[22] and C-Coupler1,[23] which adopt similar integration schemes to the above mentioned solutions, but unfortunately, they also require application modifications.

Moreover, Waston et al[24] proposed the scheme to us parallel coupling tool for effectively integrating the existing programming and performance tools, to benefit the development of parallel applications. Dorier et al[25] have summarized several tools developed by themselves, which can flexibly couple simulations with visualization packages or analysis workflows. Rivi et al[26] have explored coupling applications through their I/O interface for in situ visualization, where HDF5[27] is leveraged. GLEAN offers a flexible and extensible framework to facilitate simulation-time data analysis and I/O acceleration, when applications uses the existing I/O interfaces including HDF5.[28]

Data transfer approaches in coupling or other large-scale systems. Many integrated approaches use file-based I/O to exchange data, since the data stored on the global file system can be easily accessed by all participating components.[18] It is worth mentioning the MCT framework again, which also enables data transfer among different components via MPI communication[29] rather than file-based I/O. For instance, the CCSM4 system is a single executable implementation, which includes a top-level driver and components integrated via standard init/run/finalize interfaces by leveraging MCT.[30] From a functionality view point, the MCT tool might be the most similar approach to our work, but it requires to compile all individual models or applications together to generate a single executable binary file. The combined binary ensures that all processes can share the same MPI intracommunicator to communicate with each other through MPI function calls. However, this prerequisite is not easy to meet, because it is difficult to combine a large number of separately developed components due to possible collisions on global variables and function names.

However, since all MPI processes share the same `MPI_COMM_WORLD` communicator in MCT, local broadcast operations within a specific (individual) model becomes visible to all other processes belonging to other components. To overcome this limitation, Browne and Wilson[31] have proposed a very similar mechanism for coupling 2 specified models for the purpose of data assimilation, through a different use of the message-passing interface. In their solution, although 2 models are still compiled together to generate a single MPI job, they split the MPI communicator to enable local MPI communication within individual components. However, this solution implies that the source codes of all involved models have to be modified for enabling usage of split MPI communicators for local communication.

In addition, for supporting flexible communication patterns and better communication efficiency of I/O data transfer, the adaptable I/O system framework[32] has been proposed to support flexible direct data transfer having different I/O patterns. Similar with the OASIS coupler, the users have to modify the models or applications to use adaptable I/O system's specific interfaces. Moreover, Zhang et al have proposed and implemented a butterfly implementation of data transfer and then develop an adaptive data transfer library for the coupled systems.[33] Zhang et al presented a distributed data sharing and task execution framework to minimize interapplication data exchange.[34] Kendall et al have proposed a pattern-based I/O mechanism to boost I/O performance for large-scale parallel particle tracing and visualization applications.[35]

In summary, existing works fail to provide a general framework to integrate separately developed models or applications into a coupled system (so that direct parallel data transfer among all component models could be supported) without modifying source codes of the individual models. To the contrary of related studies, our I/O middleware intends to offer a universal communication framework to accelerate data transfer among components in coupled systems to meet strict time constraints. Additionally, our framework requires only minimal modifications to the existing application code[†].

3 | I/O ARBITRATOR MIDDLEWARE

This section discusses the design and implementation of the I/O arbitrator middleware for supporting direct parallel data transfer between the SCALE model and the LETKF model, which fulfill the simulation step and the data assimilation step, respectively, to eventually accelerate forecasting local severe weather of guerrilla rainstorms.

3.1 | Functional overview

As we mentioned before, our target coupling system of SCALE-LETKF repeats a 2-step cycle of simulation and data assimilation, performed by 2 separately developed models, ie, SCALE and LETKF. The I/O communication of one cycle in the current SCALE-LETKF system is depicted in Figure 1A. As seen, the netCDF output data of the Simulation processes are first written to the global parallel file system, which in turn is read by the Assimilation processes. To put it from another angle, in each simulation step the NWP model calculates the time evolution of atmospheric states for every grid point in 3-dimensional space, where each atmospheric state is represented as a set of variables such as the wind properties, temperature, and pressure. Note that we run multiple SCALE model simulations at the same time (denoted by Simulation 1 to *n* in the figure), which are called "ensemble" simulations and are required for applying the LETKF data assimilation scheme.

[†]In its current implementation, a single modification to the source codes for each model is required; we are implementing the version of middleware without modifications to the application through wrapping MPI initialization and finalization functions.

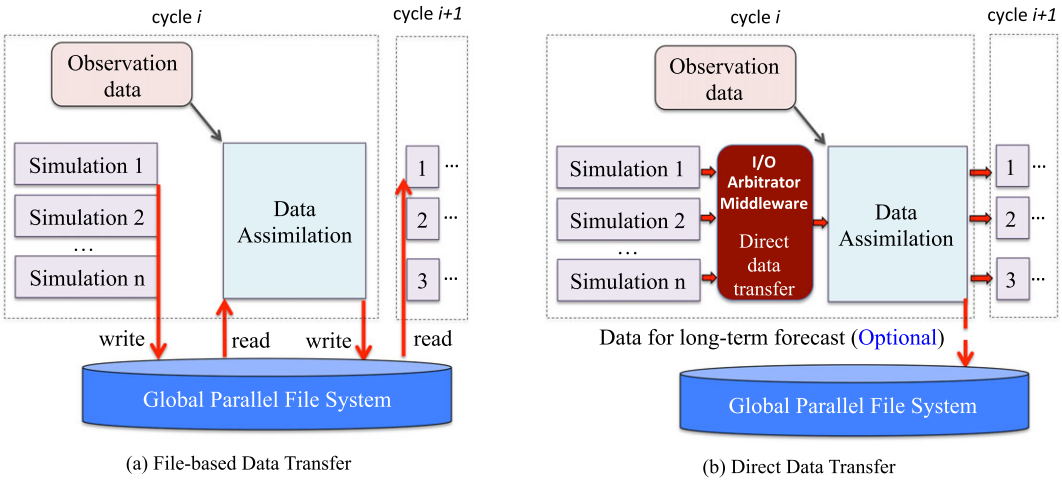


FIGURE 1 The communication pattern of one cycle in the SCALE-LETKF system using (A) file I/O or (B) direct data transfer methods

Each ensemble member takes slightly different initial condition and outputs different results so the total I/O amount is roughly equal to the I/O amount of one member multiplied by the number of ensemble members. After computation, the ensemble model of SCALE generates a large amount of output data, written in netCDF format, which are all requested by the subsequent assimilation step of the same cycle. In brief, the output data generated by the simulation process will be used by the corresponding assimilation process, which indicates that I/O communication is performed between process pairs.

According to the our tests, I/O takes 15.9% of execution time when 5769 compute nodes are used, but it consumes 37.8% of execution time while there are 37 440 compute nodes.[36] More compute nodes can decrease the time needed for computation, but they cannot benefit to the reduction of I/O time. In summary, the file I/O-based mechanism for data transfer between the data simulation and data assimilation steps must place negative effects on real-time predicting for local severe weather of guerrilla rainstorms.

To reduce the time needed for data transfer, we have been developing a novel I/O middleware to allow direct parallel data transfer between the 2 component models. Figure 1B illustrates the workflow of the system when the I/O middleware is used. As a result, in each cycle, the output data of simulation processes are directly forwarded to the assimilation processes, as well as the analyzed results generated by assimilation processes, which can be directly transferred to the simulation processes in the next cycle. Specifically, the I/O middleware connects the 2 kinds of processes by using MPI communication,[29] and consequently, it enables direct communication between the simulation processes and the assimilation processes. The following section describes the details of establishing a communication environment between the 2 types of processes.

Note that our proposed I/O middleware is a general solution for coupled Big Data processing applications although only the SCALE-LETKF application is detailed in this paper. To handle a wide range of possible I/O patterns, the middleware is customizable using configurations files. Different configurations enable deployment for applications with different properties, such as different number of component models, or different I/O communication patterns. The role of configuration file will be further described in Section 3.6.

3.2 | High level architecture

Figure 2 shows the I/O stack of the I/O arbitrator middleware, which is used to support direct parallel data transfer between simulation processes (SIM in the figure) and data assimilation processes (DA in the figure) in our case study. Except for the application layer itself, the layer of netCDF, the layer of POSIX, and the layer of MPI have been involved in the middleware for the purpose of sustaining direct parallel data transfer. Briefly speaking, the newly proposed middleware is built underneath the layer of the application; the mechanism of direct parallel data transfer is therefore transparent to the applications.

Besides, as shown in the figure, the data communication will be completed by using the MPI communication facility, and the following subsection will discuss the details of constructing the communication context between 2 kinds of processes.

3.3 | Establishing communication

Because the simulation and data assimilation models are separately developed applications and are executed as separate MPI jobs, they do not share the same MPI communicator. To overcome this problem, our prototype implementation currently uses the standard MPI intercommunicator family of routines to establish a communication context between the 2 types of jobs. Figure 3 demonstrates the details of this approach. Note that the K computer, our target platform in this paper, supports the creation of intercommunicators among jobs submitted by the same user. However, we emphasize that our MPI-based implementation is merely a proof of concept and in the long run, we are planning to leverage a low-level communication (LLC) framework,[37] which is currently being developed as part of the post-K project.

Nevertheless, we provide an overview of the current MPI-based implementation. At initialization time, the server process, ie, an Assimilation process opens a port using `MPI_Open_port()` and then publishes it by calling the `MPI_Publish_name()` feature. Subsequently, the connection thread of each individual Assimilation process waits in `MPI_Comm_accept()`. The connection service is expected to be already running by the time, when the client processes attempt to build the connection. Client processes, ie, processes of

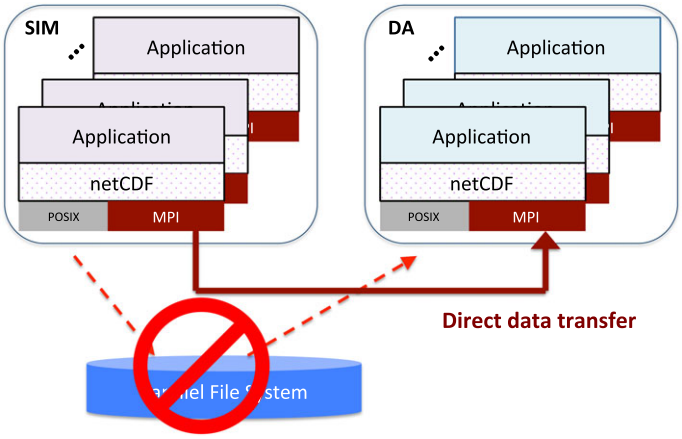


FIGURE 2 Architectural overview of the middleware

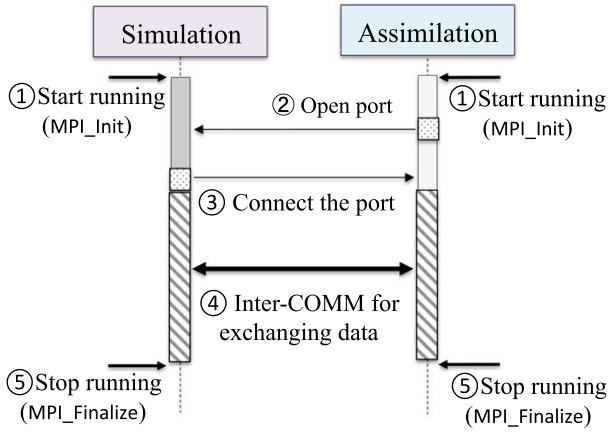


FIGURE 3 Establishing communication between workflow components

the Simulation component, can connect to the server processes with `MPI_Comm_connect()` once they successfully obtained the service name by using the `MPI_lookup_name()` function. As a result, processes of both components can communicate with each other by using standard MPI functions. For instance, the Simulation processes can use the `MPI_Send()` function to forward the data, and the Simulation processes are able to use `MPI_Recv()` to receive the data. After the data transfer took place, the client processes proactively disconnect and the server processes can unpublish their connection services with `MPI_Unpublish_name()`.

1. The Simulation process attempts to write the output data to the file system through calling the `write()` system call. We assume that the Assimilation process will eventually read the contents of the same file, but the Assimilation process is supposed to be blocked until the requested data is satisfied in Step ⑥.
2. The `write()` call can be eventually satisfied by the `px_pgout()` function in the netCDF library, which is intercepted by library hook offered by the middleware. Then the write contents are cached in the designated memory buffer, instead of flushing them to the global file system.
3. The buffered data is forwarded from the Simulation node to the destination node, ie, the Assimilation process, by calling the `MPI_Send()` routine.
4. The Assimilation process responds an ACK message, when it has received the data sent by the corresponding Simulation process, through calling `MPI_Recv()`. Consequently, the data is cached in the designated memory buffer for satisfying potential future read requests.
5. According to the parameters of the `read()` request (it goes to the function of `px_pgin` in the netCDF library), which was blocked because the required data were not yet available, the specified piece of data will be picked up by library hook from memory buffer.
6. The Assimilation process resumes its execution after it received the data from library hook.

3.4 | Parallel direct data transfer

The parallel data transfer between the simulation processes and the assimilation processes is conducted when the communication context has been constructed. Figure 4 depicts the details of the parallel direct data transfer in the I/O middleware, where the interaction between 2 kinds of processes can be described as follows:

Both Simulation and Assimilation processes are able to exchange their data through direct data transfer. Specifically, all `write()` requests will be fulfilled when the contents have been buffered in the memory, and all cached data are eventually sent to the destination process. On the other side, all `read()` requests will be satisfied with the data buffered in the memory, which was initially received from the source process.

We provide 2 kinds of direct data transfer schemes, ie, synchronous and asynchronous data transfer, which have been illustrated in Figure 5. In synchronous data transfer, all data are transferred when the output data have been completely flushed in to the memory buffer (ie, the output netCDF file is closed). On the other side, the idea of our asynchronous transfer scheme is to split the computation into multiple steps and then execute computation and data transfer in parallel. Explicitly, the mechanism of asynchronous data transfer uses a communication thread, ie, Comm. Thread in Figure 5B, to aperiodically send a part of the flushed data (in the memory buffer) from one side to another side, during the computation. Then, the computation and data transfers can be processed in parallel. At last, only the dirty data, which have been updated since the previous data transmission, are transferred again,

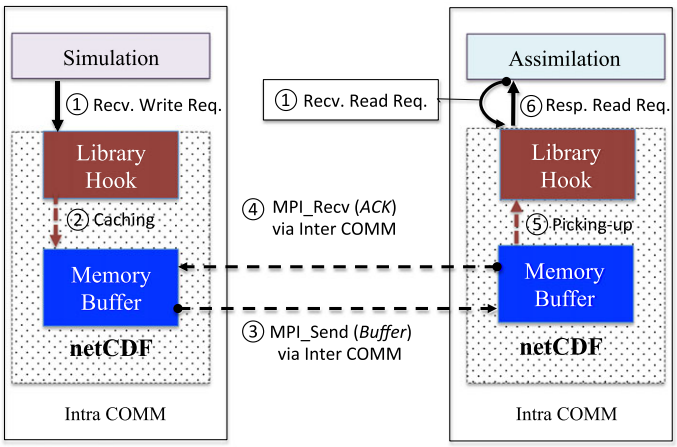
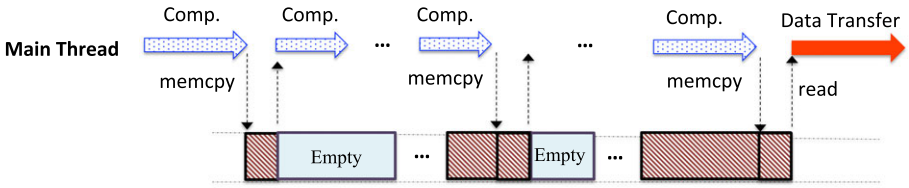
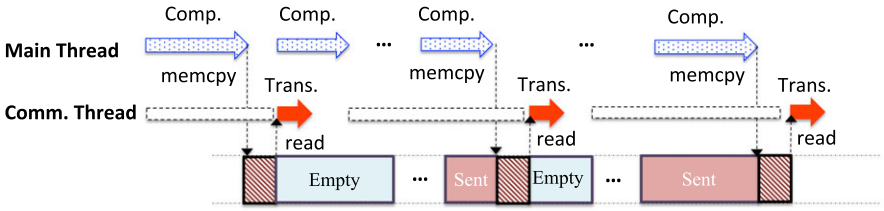


FIGURE 4 Synchronized direct data transfer among workflow components



(a) Synchronous Data Transfer



(b) Asynchronous Data Transfer

FIGURE 5 Synchronized and asynchronous data transfer (both of them are supported by the proposed middleware)

after the output netCDF file is closed. We call the time required for sending the remaining data and the dirty data after all computation tasks of the sender sides, as the observed communication time in this paper (the last red arrow in Figure 5A,B).

3.5 | Communication pattern support

The proposed middleware supports various communication patterns, ie, data can be sent to the appropriate destination according to application requirements. Similarly to the MPI library, the middleware provides collective communication routines, including scatter/gather and all-to-all patterns, to enable various patterns of data transfer in the target applications.

Note that the selection of the actual communication routine is performed by the middleware, and users only need to set the configuration file to specify the communication pattern among the component modules before running the application. A detailed description of the configuration file will be provided in the following subsection.

3.6 | Customizability via configuration file

To retain full application transparency, we propose a scheme of configurable transfer settings, which allow users to specify the components involved in the application, along with the files that are denoted to be transferred from a given component to another.

The configuration file offered by the framework requires specifying the followings. Users need to indicate each individual workflow component (currently identified by executable names), information about the connection between components (ie, the connection name), details of the files that will bypass the parallel file system (denoted by file names that also support asterisk-based name completion), and the actual communication pattern (such as, one-to-one, one-to-many, etc).

Figure 6 demonstrates a simplified configuration example in SCALE-LETKF. As seen, the component COMP0 outputs the file called "history," which in turn is required to be sent to COMP1 as an input file. The name of the connection (used for establishing the MPI intercommunicator connection) is called "scale-obs," where the communication pattern is indicated to be a one-to-one type of data transfer.

Given that an application links against our library, the configuration setting is loaded during initialization, and the specified settings will guide the direct data transfer mechanism during execution.

Although in the case study of SCALE-LETKF, only the one-to-one (pair-based) communication pattern is leveraged, in which a particular process sends data to exactly one corresponding process, it is also

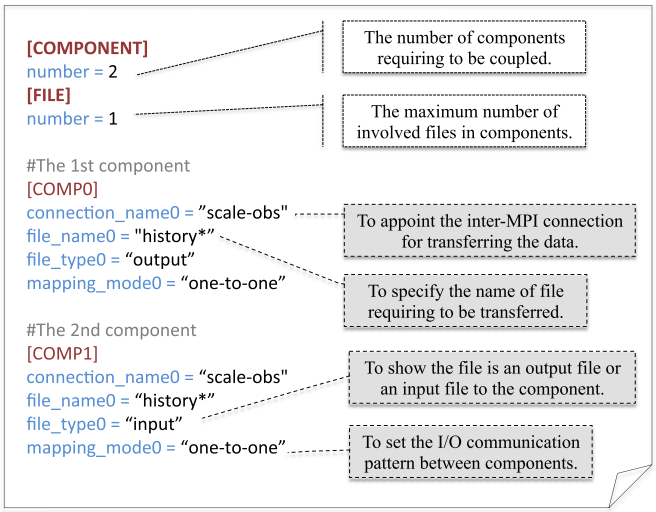


FIGURE 6 A simplified configuration example of the direct I/O middleware

possible to set the configuration file to enable transferring the data by following one-to-many or many-to-many mappings. Moreover, the all-to-all communication functionality is also supported in the current implementation of our middleware. As a result, complex communication patterns can be performed entirely by the middleware incorporating data redistribution that would have to be done at application level otherwise.

3.7 | Implementation for SCALE-LETKF

For demonstrating the effectiveness of direct parallel data transfer between the simulation and assimilation processes in SCALE-LETKF, we have developed a proof-of-concept implementation of the proposed I/O middleware. In addition, since data is exchanged between each SCALE process and the corresponding LETKF process in netCDF format, we have made slight modifications to the netCDF library itself (using ver. 4.2.2.1), so that it complies with the proposed I/O middleware to enable direct data transfer in an application transparent fashion.

4 | EVALUATION

This section first describes the experimental setup and experimental methodology for evaluating the proposed I/O middleware. It then presents experimental results and provides the relevant discussion. At last, we summarize the key points of our direct parallel data transfer approach.

4.1 | Experimental setup

Evaluation experiments to assess the advantages of the SCALE-LETKF system equipped with our current prototype middleware were conducted on the K computer.[38] The K computer is Japan's flagship supercomputer sporting 88 128 compute nodes (8 CPU cores each), with peak performance more than 10 petaFLOPS. The K computer took the first place of TOP 500 in 2011, and as of June 2016, it is ranked as the fifth fastest machine of the world.[39] Table 1 gives an overview specification of the K computer.

As for the input data used in our experiments, we use real-world observations to test the efficiency of SCALE-LETKF when equipped with the proposed I/O middleware. In all experiments, each MPI process was allocated to one compute node, and we logged the results related to I/O operations during the execution. Note that every MPI process is allocated onto one computing node, and openMP is used to explicitly direct multithreaded parallelism.

Figure 7 demonstrates the I/O workflow of one cycle in the application. The LETKF component actually contains the modules of OBSOPE and LETKF, both of which need to obtain data from SCALE. In one-cycle of the execution, the SCALE module outputs 2 kinds of results, the "history" data and the "restart" data, which are read by OBSOPE and LETKF modules, respectively. In addition, the OBSOPE is responsible for generating the analyzed data that is requested by the LETKF module.

Three real-world test cases for regional weather analysis were used. In each measurement, SCALE is composed of up to 100 ensemble instances. test case 1 and Test Case 2 have 4 processes in each ensemble instance, but there are 48 processes in each ensemble instance of test case 3. LETKF consists of only one instance, but it contains the same number of processes as all SCALE instances in total.

TABLE 1 Specification of the K computer

Node specification	CPU Performance Memory	SPARC64 VIIIfx 2GHz 128 GF (16 GF × 8 cores) 16 GB
Number of racks		**864**
Number of nodes		**82, 944**
Network		**Tofu 6D Mesh/Torus**
Link bandwidth		**5 GB/s × bidirectional**
Peak performance		**10.62 petaFLOPS**
Total memory capacity		**1.26 PB**
Global file system		**Fujitsu Exabyte File System**
Storage capacity		**30 PB**

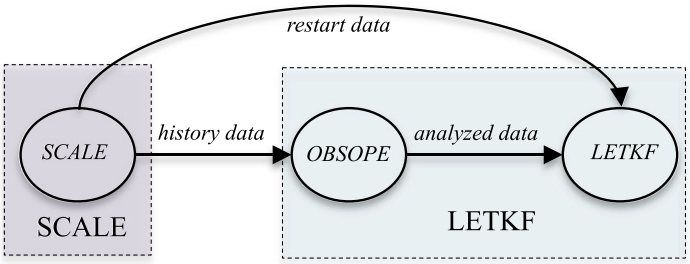


FIGURE 7 I/O workflow in SCALE-LETKF in one cycle: all transferred data are organized with netCDF format

- Test case 1: testcase_45km_4p_l36:
  - Number of variables at one gird: 11
  - Ensemble size: varying from 10 to 100
  - Total variable number: 3.2×10^8
  - Total I/O size: up to 34 GB for the entire 100 ensemble members
- Test case 2: testcase_45km_4p_l72:
  - Number of variables at one gird: 11
  - Ensemble size: varying from 10 to 100
  - Total variable number: 6.2×10^8
  - Total I/O size: up to 67 GB for the entire 100 ensemble members
- Test case 3: testcase_15km_48p_l36:
  - Number of variables at one gird: 11
  - Ensemble size: varying from 10 to 100
  - Total variable number: 4.4×10^8
  - Total I/O size: up to 530 GB for the entire 100 ensemble members

Table 2 summarizes the size of transferred data for the cases having different number of ensemble instances. Note that in our current execution model, each application instance corresponds to a separate MPI job.

4.2 | Experimental results

The main limitation of our current proof-of-concept I/O middleware is that we can run only one cycle of the SCALE-LETKF system, because SCALE cannot run multiple cycles. In other words, each SCALE process generates output data after simulation, which will be read by the corresponding LETKF process as input for assimilation.

4.2.1 | Inter-job MPI bandwidth

As we mentioned before, our current prototype implementation uses MPI communication for building connection between components, as well as for enabling direct data transfer. Because high-end HPC platforms do not necessary support communication between separate MPI jobs efficiently, we leveraged a modified version of the *osu_bw* microbenchmark from the MVAPICH suite[40] to verify that intra- and inter-job MPI communication on the K computer indeed yields the same performance. The experiments were performed by exchanging a number of ping-pong messages between the Simulation and the Assimilation processes after the 2 components have been connected by the middleware.

TABLE 2 Total amount of transferred data in the case study

Ensemble size	Test case 1	Test case 2	Test case 3
10	3468 MB	6720 MB	53 328 MB
20	6936 MB	13 440 MB	106 656 MB
40	13 872 MB	26 880 MB	213 312 MB
60	20 808 MB	40 320 MB	319 968 MB
80	27 744 MB	53 760 MB	426 624 MB
100	34 680 MB	67 200 MB	533 280 MB

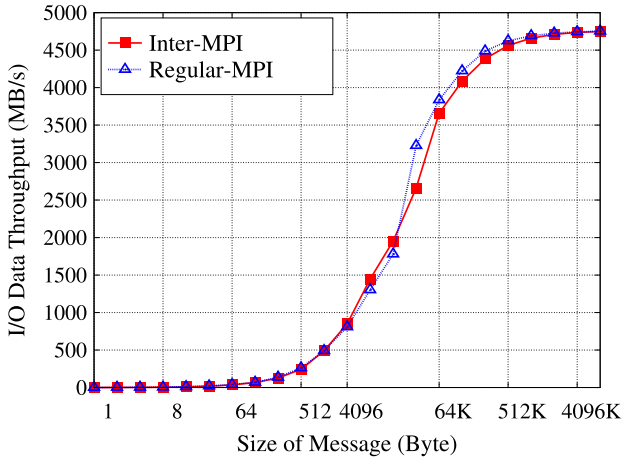


FIGURE 8 Intra- and inter-job MPI communication bandwidth

Figure 8 demonstrates the experimental results of the network bandwidth, where the horizontal axis represents message size and the vertical axis indicates the bandwidth. As seen, the attained bandwidth by 2 communication schemes do not appear to be noticeable different, which indicates that the inter-job MPI communication scheme used by the middleware does not place any negative effects in our target environment.

4.2.2 | Communication time

While running the selected 3 test cases, we first measured the observed communication time for transferring data between SCALE and LETKF after the computation of SCALE, as the function of increasing the number of ensemble instances from 10 to 100. Figure 9 shows the time required for transferring the data from SCALE to LETKF by using 2 direct data transfer schemes. The horizontal axis represents the number of ensemble instances. As mentioned, every ensemble instance in both test cases 1 and 2 has 4 processes, and each ensemble instance in test case 3 has 48 processes. The vertical axis shows the time required for data transmission. As the experimental results imply, the communication time for transferring data between the 2 components remains essentially unchanged, even with the growing number of

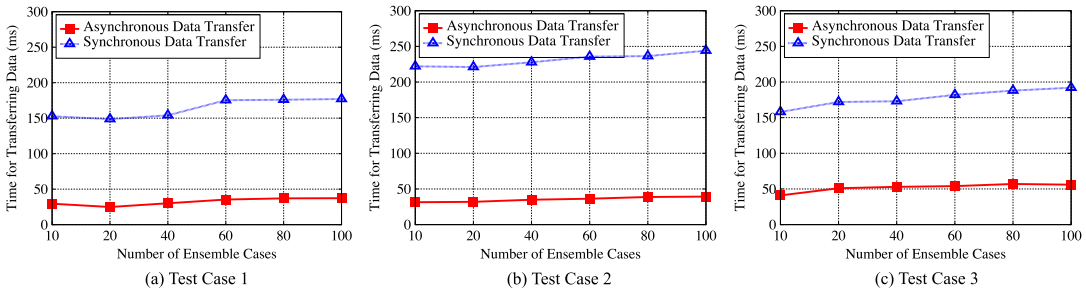


FIGURE 9 Observed communication time needed for transferring data from SCALE to LETKF after computation of SCALE (both of them are supported by the proposed middleware)

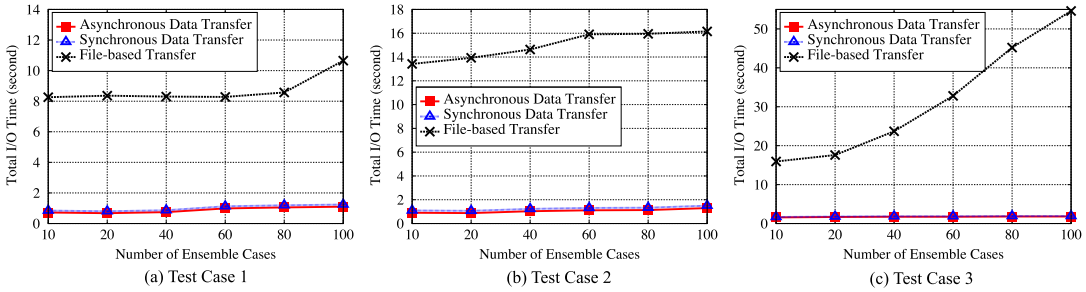


FIGURE 10 I/O time contrasting file I/O and direct data transfer having synchronous transfer and asynchronous transfer schemes

involved processes, which is due to the pair-wise communication pattern of the SCALE-LETKF system.

Another interesting observation is that the asynchronous transfer scheme requires less than `one-third` time for transferring the data after the computation of data simulation, compared with synchronous data transfer. In other words, the LETKF processes can start the computation of data assimilation quite earlier, when we use the asynchronous transfer scheme. This is because a major part of output data has been concurrently transferred with the computation in the main thread, and the dirty data occupy a small proportion of all output data.

4.2.3 | I/O acceleration

For comparison, we recorded the time required for I/O operations between the SCALE and LETKF processes by using both actual file I/O operations and the mechanism of direct data transfer. Figure 10A-C indicates the time required for conducting I/O operations between the 2 types of processes using file I/O and the proposed mechanism, respectively. Note that the I/O time shown by the proposed mechanisms includes the time needed for memory operations, and the time required for transferring the data from SCALE to LETKF.

As the Figure depicts, the proposed mechanism can substantially reduce the time needed for I/O operations between SCALE and LETKF processes compared to the file I/O-based data transfer. For example, when the size of ensemble instances is 100 using the case of test case 3, the mechanism of direct data transfer can yield over 30× speedup on I/O operations, which in turn implies that more time can be devoted to perform simulation and data assimilation and that the total execution time can be consequently decreased. Furthermore, the file-based data transfer may require significantly increased I/O time due to contention on the parallel file system. The case of test case 2 required 34.1% more time for conducting file I/O operations, compared with the case of test case 1, because the size of transmission data needed by the former case is 2 times of the size of transmission data of the latter one. In contrast, direct data transfer does not increase the transfer time significantly even for double size data.

We also report a breakdown analysis of the time required for I/O operations among the component models in SCALE-LETKF. Figure 11 indicates the results in detail. To be precise, the LETKF model has 2 modules including OBSOPE and LETKF, both of which request input data from SCALE (previously indicated by Figure 7). The experimental results clearly show that all components consume less time for completing I/O operations while using the proposed I/O middleware, which is the reason we could decrease execution time in total.

4.2.4 | Data throughput

After verifying, the proposed mechanism can indeed reduce the time needed for exchanging the data between SCALE and LETKF in our test cases, this section aims to measure the I/O data throughput while executing various test cases. Figure 12A-C shows the results about I/O data throughput reported by performing the tests with varying ensemble sizes, respectively. As seen, the proposed scheme of direct data transfer outperforms the scheme of file I/O-based data transfer, and it achieves from 758.3% to 2933.3% data rate improvements for the selected test cases. Particularly, improvements are getting remarkable while the ensemble size is getting larger that indicates more data are required to be processed.

Another remarkable issue, implied by the figures, is the fact that the larger the amount of data is to be exchanged, the higher the benefits become by using the direct data transfer method. In addition, compared with the scheme of *Synchronous Data Transfer*, the *Asynchronous Data Transfer* scheme can achieve more attractive performance improvements.

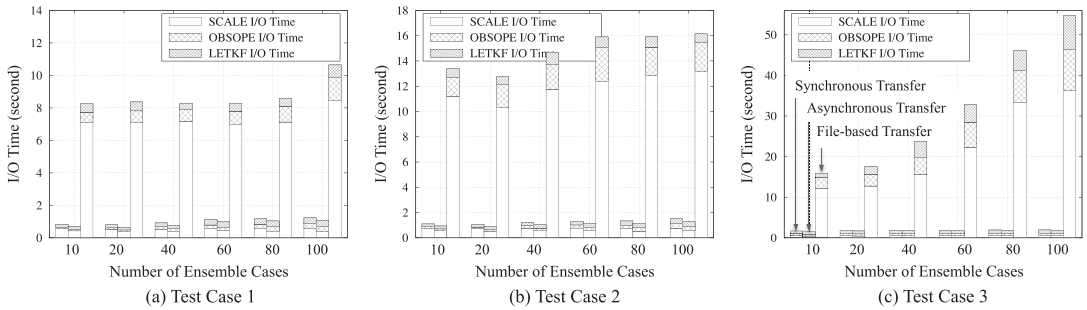


FIGURE 11 I/O time breakdown comparing file I/O and direct data transfer having synchronous transfer and asynchronous transfer schemes

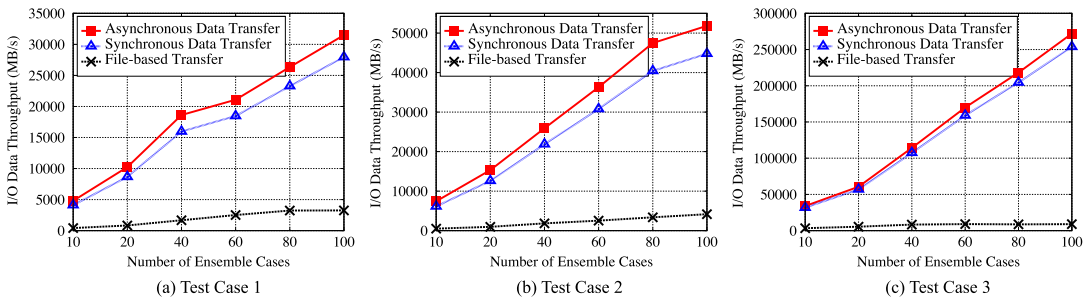


FIGURE 12 I/O data throughput using file I/O and direct data transfer having synchronous transfer and asynchronous transfer schemes

4.3 | Summary

With respect to comparing direct data transfer and file I/O based data transfer, we emphasize the following 2 key observations. First, with increasing number of processes, direct data transfer yields better relative performance. Second, the more time reduction and higher data throughput can be achieved with the growing size of the involved data. In brief, we conclude that the proposed file I/O middleware is able to significantly reduce the time required by exchanging data between the component models in the *SCALE-LETKF* workflow system.

Furthermore, the implemented I/O middleware offers a general framework for inter-component data exchange in workflow systems, where individually developed applications are coupled together. By accelerating the execution of such systems, we believe our newly proposed middleware, facilitated with the direct data transfer functionality, is particularly important for systems with rigorous time constraints.

5 | CONCLUDING REMARKS

This paper has proposed a general I/O middle for Big Data processing coupled workflows that are comprised of multiple individually developed components. Most importantly, it enables direct parallel data transfer among the component models for reducing the time required for data exchange among them. We have applied this I/O middleware to the *SCALE-LETKF* data assimilation based weather forecasting system for allowing data transfer via inter-MPI communication between simulation and data assimilation processes, to forecast local severe weather of guerrilla rainstorms with acceptable prediction period.

In order to put this mechanism to work, we first build a communication context among the different types of processes by using the inter-MPI communication facility. Then a library hook built in the *netCDF* library is responsible for intercepting all I/O (i.e., read/write) requests until the needed data has been transmitted to the corresponding process. Consequently, compared to the mechanism of file I/O based data exchange, the time required for I/O operations can be significantly reduced, and the data throughput is greatly improved. Experimental results have demonstrated that the proposed direct parallel data transfer can reduce the time needed for I/O operations among the SCALE and LETKF processes by between 65.6% and 92.3%. Our tests also illustrate the direct parallel data transfer could increase the I/O data throughput by between 758.3% and 2933.3%, in contrast to file I/O-based transfer.

Especially, experimental results on the K computer using up to 4800 nodes have shown that the proposed mechanism can significantly reduce the time spent on I/O operations among SCALE and LETKF. This achievement is useful for real-time weather forecasting in SCALE-LETKF or similar applications, because the I/O time does not noticeably increase while the problem scale is getting larger. Therefore, we conclude that the most attractive results yielded by the newly proposed I/O middleware is that the benefits of larger data throughput increase with the growing amount of data that are required to be processed.

The current implementation of the middleware relies on the MPI library for data transmission, but our long-term vision is to implement data transfer on top of a LLC, which will enable us to establish connections among arbitrary MPI jobs. Low level communication is part of the development plan of the post-K supercomputer (the next generation flagship supercomputer in Japan). In addition, enabling asynchronous data transfer so that communication and computation can be efficiently overlapped is another important item on the list of our future work. Furthermore, with asynchronous data transmission, we also intend to explore the use of direct RDMA operations between individual components so that any unnecessary buffering can be eliminated during the data exchange.

ACKNOWLEDGMENTS

This work has been partially supported by CREST, JST, and the MEXTs program for the Development and Improvement of Next Generation Ultra High-Speed Computer Systems, National Natural Science Foundation of China (No. 61303038), and the Opening Project of State Key Laboratory for Novel Software Technology (No. KFKT2016B05). This research used computational resources of the K computer provided by the RIKEN Advanced Institute for Computational Science through the HPCI System Research project (Project ID: hp150019). The authors would like to thank the Euro-par 2016 anonymous reviewers for their thorough reviews and highly appreciate the comments and suggestions, which significantly contributed to revise this paper.

REFERENCES

1. Reed D, Dongarra J. Exascale computing and big data. *Commun ACM*. 2015;58:56–68.
2. Lien GY, Miyoshi T, Nishizawa S, Yashiro H, Yoshida R, Tomita H. Ensemble data assimilation for a large parallel numerical weather prediction model: Development of the SCALELETKF system. *Proceedings of the 2015 International Symposium on Data Assimilation*, ISDA '2015. Kobe, Japan; February 2015.
3. Chen G, Sawada M, Sha W, Saito K, et al. A challenge to realize the ultra-high-resolution weather forecast for big city. *Proceeding of Symposium on K-computer*. Kobe, Japan; 2012:71–72.
4. Miyoshi T, Kondo K, Terasaki K. Big ensemble data assimilation in numerical weather prediction. *IEEE Computer*. 2015;48(11): 15–21.
5. Sato Y, Nishizawa S, Yashiro H, et al. Impacts of cloud microphysics on trade wind cumulus: which cloud microphysics processes contribute to the diversity in a large eddy simulation?*Progress Earth Planet Sci*. 2015;2(1): 1–16.
6. Nishizawa S, Yashiro H, Sato Y, Miyamoto Y, Tomita H. Influence of grid aspect ratio on planetary boundary layer turbulence in large-eddy simulations. *Geoscientific Model Dev*; 8(10): 3393–3419.
7. Hunt B, Kostelich E, Szunyogh I. Efficient data assimilation for spatiotemporal chaos: a local ensemble transform Kalman filter. *Phys D: Nonlinear Phenom*. 2007;230(1): 112–126.
8. Network Common Data Form (netCDF). Available from: http://www.unidata.ucar.edu/software/netcdf/ (Accessed on Oct., 2015).
9. Chen F, Dudhia J. Coupling an advanced land surface-hydrology model with the Penn State-NCAR MM5 modeling system. *Part I: Model Implementation and Sensitivity. Mon Weather Rev*. 2001;129(4): 569–585.
10. Chen F, Kusaka H, Bornstein R, et al. The integrated WRF/urban modelling system: development, evaluation, and applications to urban environmental problems. *Int J Climatol*. 2011;31(2): 273–288.
11. Niu G, Paniconi C, Troch A, et al. An integrated modelling framework of catchmentscale ecohydrological processes: 1. Model description and tests over an energylimited watershed. *Ecohydrol*. 2014;7(2): 427–439.
12. Linares-Rodriguez A, Lara-Fanego V, Pozo-Vazquez D, et al. One-day-ahead streamflow forecasting using artificial neural networks and a meteorological mesoscale model. *J Hydrol Eng*. 2015;20(9):05015001, 9 pages.
13. Larson J, Jacob R, Ong E. The Model Coupling Toolkit: a new Fortran90 toolkit for building multiphysics parallel coupled models. *Int J High Perform Comput Appl*. 2005;19(3): 277–292.
14. Craig AP, Jacob R, Kauffman B, He Y, et al. CPL6: The New extensible, high performance parallel coupler for the Community Climate System Model. *Int J High Perform Comput Appl*. 2005;19(3): 309–327.
15. Valcke S, Craig A, Dunlap R, Riley G. Sharing experiences and outlook on coupling technologies for earth system models. *Bull Amer Meteor Soc*. 2015;97(3):3, ES53–ES56. https://doi.org/10.1175/BAMS-D-15-00239.1
16. Liao J, Gerofi B, Lien GY, et al. Toward a general I/O arbitration framework for netCDF based big data processing. *Proceedings of the 22nd International Conference on Parallel and Distributed Computing*, (Euro-Par '2016). Grenoble, France; August 2016:293–305.
17. Ji Y, Zhang Y, Yang G. Interpolation oriented parallel communication to optimize coupling in earth system modeling. *Front Comput Sci*. 2014;8(4): 693–708.
18. Valcke S, Balaji V, Craig A, Riley G, et al. Coupling technologies for earth system modelling geoscientific model development; 2012.
19. Valcke S. The OASIS3 coupler: a European climate modelling community software. *Geoscientific Model Development Discussions*. 2013;6(2):373–388.
20. Armstrong CW, Ford RW, Riley GD. Coupling integrated earth system model components with BFG2. *Concurrency Comput: Pract Exp*. 2009;21(6): 767–791.
21. Janjic Z., Black T. An ESMF unified model for a broad range of spatial and temporal scales. *Geophys Res Abstr*. 2007;9:05025.
22. Balaji V, Anderson J, Held I, et al. The FMS Exchange Grid: A mechanism for data exchange between Earth System components on independent grids; 2007.
23. Liu L, Yang G, Wang B, et al. C-coupler1: A Chinese community coupler for Earth system modelling. *Geosci Model Dev Discuss*. 2014;7(3): 3889–3936.
24. Watson G, Frings W, Knobloch C, et al. Scalable control and monitoring of supercomputer applications using an integrated tool framework. *Proceedings ICPPW '2011*. Taipei, Taiwan, China; 2011:457–466.
25. Dorier M, Dreher M, Peterka T, et al. Lessons learned from building in situ coupling frameworks. *Proceedings of the First Workshop on in Situ Infrastructures for Enabling Extreme-Scale Analysis and Visualization*. Austin, TX, USA: ACM; 2015:19–24.
26. Rivi M, Calori L, Muscianisi G, et al. In-situ visualization: State-of-the-art and some use cases. PRACE White Paper; PRACE, Brussels, Belgium; 2012.
27. HDF5. Available from: https://support.hdfgroup.org/HDF5/ (Accessed on Oct., 2016).
28. Vishwanath V, Bui H, Hereld M, Papka M. *GLEAN*, High Performance Parallel I/O. Florida, USA: CRC Press, Taylor and Francis Group; November 2014.
29. Message Passing Interface Forum. MPI: A Message-Passing Interface Standard, Version 2.2; September 2009.
30. Craig A, Vertenstein M, Jacob R. A new flex- ible coupler for earth system modeling developed for CCSM4 and CESM1. *Int J High Perform C*. 2012;26(1): 31–42.
31. Browne PA, Wilson S. A simple method for integrating a complex model into an ensemble data assimilation system using MPI. *Environ Model Softw*. 2015;68:122–128.
32. Podhorszki N, Klasky S, Liu Q, et al. Plasma fusion code coupling using scalable I/O services and scientific workflows. *Proceedings of the 4th Workshop on Workflows in Support of Large-Scale Science*, WORKS '2009. Portland, Oregon, USA; November 2009.
33. Zhang C, Liu L, Yang G, et al. Improving data transfer for model coupling. *Geosci Model Dev Discuss*. 2015;8:8981–9020.
34. Zhang F, Docan C, Parashar M, et al. Enabling in-situ execution of coupled scientific workflow on multi-core platform. *Proceedings of IEEE 26th International Parallel & Distributed Processing Symposium*, IPDPS '2012. Shanghai, China; 2012:1352–1363.
35. Kendall W, Huang J, Peterka T, Latham R, Ross R. Toward a general I/O layer for parallel-visualization applications. *IEEE Comput Graphics Appl*. 2011;31(6): 6–10.
36. Ishikawa Y, Choudary A, Liao Wk, Liao J, Gerofi B. The presentation materials (informal publication). *The 3rd Meeting on Japan-US DOE-MEXT Collaboration for HPC System Software*. Tokyo, Japan; February 2016.

37. Takagi M, Yamaguchi N, Gerofi B, Hori A, Ishikawa Y. Adaptive transport service selection for MPI with InfiniBand network. *Proceedings of the 3rd Workshop on Exascale MPI*. Austin, TX, USA; 2015:3.
38. RIKEN AICS: K computer. Available from: http://www.aics.riken.jp/en/k-computer/ (Accessed on June, 2016).
39. TOP500 Supercomputer Sites. Available from: http://www.top500.org/ (Accessed on Nov., 2016).
40. OSU Micro-benchmarks. Available from: http://mvapich.cse.ohio-state.edu/benchmarks/ (Accessed on July, 2013).

How to cite this article: Liao J, Gerofi B, Lien G-Y, et al. A flexible I/O arbitration framework for netCDF based big data processing workflows on high-end supercomputers. *Concurrency Computat: Pract Exper*. 2017;29:e4161. https://doi.org/10.1002/cpe.4161