

Potential of I/O aware workflows in climate and weather

Article

Published Version

Creative Commons: Attribution-Noncommercial 3.0

Open Access

Kunkel, J. M. and Pedro, L. R. (2020) Potential of I/O aware workflows in climate and weather. *Supercomputing Frontiers and Innovations*, 7 (2). pp. 35-53. ISSN 2313-8734 doi: <https://doi.org/10.14529/jsfi200203> Available at <https://centaur.reading.ac.uk/92436/>

It is advisable to refer to the publisher's version if you intend to cite from the work. See [Guidance on citing](#).

Published version at: <http://dx.doi.org/10.14529/jsfi200203>

Identification Number/DOI: <https://doi.org/10.14529/jsfi200203>

<<https://doi.org/10.14529/jsfi200203>>

Publisher: South Urals University

All outputs in CentAUR are protected by Intellectual Property Rights law, including copyright law. Copyright and IPR is retained by the creators or other copyright holders. Terms and conditions for use of this material are defined in the [End User Agreement](#).

www.reading.ac.uk/centaur

CentAUR

Central Archive at the University of Reading

Reading's research outputs online

Potential of I/O Aware Workflows in Climate and Weather

*Julian M. Kunkel*¹ , *Luciana R. Pedro*¹ 

© The Authors 2020. This paper is published with open access at SuperFri.org

The efficient, convenient, and robust execution of data-driven workflows and enhanced data management are essential for productivity in scientific computing. In HPC, the concerns of storage and computing are traditionally separated and optimised independently from each other and the needs of the end-to-end user. However, in complex workflows, this is becoming problematic. These problems are particularly acute in climate and weather workflows, which as well as becoming increasingly complex and exploiting deep storage hierarchies, can involve multiple data centres.

The key contributions of this paper are: 1) A sketch of a vision for an integrated data-driven approach, with a discussion of the associated challenges and implications, and 2) An architecture and roadmap consistent with this vision that would allow a seamless integration into current climate and weather workflows as it utilises versions of existing tools (ESDM, Cylc, XIOS, and DDN's IME).

The vision proposed here is built on the belief that workflows composed of data, computing, and communication-intensive tasks should drive interfaces and hardware configurations to better support the programming models. When delivered, this work will increase the opportunity for smarter scheduling of computing by considering storage in heterogeneous storage systems. We illustrate the performance-impact on an example workload using a model built on measured performance data using ESDM at DKRZ.

Keywords: workflow, heterogeneous storage, data-driven, climate/weather.

Introduction

High-Performance Computing (HPC) harnesses the fastest available hardware components to enable the execution of tightly coupled applications from science and industry. Typical use-cases include numerical simulation of physical systems and analysis of large-scale observational data. In the domain of climate and weather, there is a considerable demand for the orchestration of ensembles of simulation models and the generation of data products. A service such as the operational weather forecast workflow in Met Office writes around 200 TB and reads around 600 TB every day. In total, at the Met Office, on average 1.5 PB and 14 PB are written and read per day, respectively, for all climate and weather forecasts across all HPC clusters.

Based on the needs of climate and weather researchers, the HPC community has developed a software ecosystem that supports scientists to execute their large-scale workflows. While the current advances correspond to a big leap forward, many processes still require experts. For example, porting a workflow from one system to another requires adjusting runtime parameters of applications and deciding on how data is managed.

Since performance is of crucial importance to large-scale workflows, careful attention must be paid to exploit the system characteristics of the target computing centre. For instance, a data-driven workflow may benefit from the explicit and simultaneous use of a locally heterogeneous set of computing and storage technologies. This aspect means that substantial changes may be required to a workflow to tailor it to a particular supercomputer environment in order to obtain the best performance.

Knowing the capabilities, interfaces, and performance characteristics of individual components are mandatory to make the best use of them. As the complexity of systems expands and alternative storage and computing technologies provide unique characteristics, it becomes in-

¹University of Reading, Reading, United Kingdom

creasingly difficult, even for experts, to manually optimise the usage of resources in workflows. In many cases, modifications are not performed because: 1) They are labour intense: any change to the workflow requires careful validation which may not pay off for small scale runs; 2) It is a one-time explorative workflow and; 3) Users are not aware of the potential of the complex system.

In this paper, we illustrate how knowing the Input/Output (I/O) characteristics of workflow tasks and overall experimental design helps to optimise the execution of climate and weather workflows. Exploiting this information automatically may increase the performance, throughput and cost-efficiency of the systems, providing an incentive to users and data-centres that cannot be neglected any longer. Our approach intends to reduce the burden on researchers and, at the same time, optimise the decisions about jobs running on HPC systems.

This paper is structured as follows. First, we describe the software stack involved in executing workflows in climate and weather in Section 1. Related work in heterogeneous storage environments and solutions for workflow processing is presented in Section 2. Next, the vision for including knowledge about data requirements and characteristics is sketched in Section 3 outlining the potential benefit the automatic exploitation might bring. Our design, based on existing components in climate and weather, is described in Section 4. An example use case demonstrates the impact on running a workload at the Mistral supercomputer in Section 5. The paper is concluded in Section 5.

1. Workflows in Climate/Weather

In this section, we describe how workflows are executed in a conventional software stack and the typical hardware and software environment involved in running a climate and weather application.

1.1. Cylc

Cylc [19] is a general-purpose workflow engine in charge of executing and monitoring cyclic workflows in which each step is submitted to the batch scheduler of a data centre. With Cylc, tasks from multiple cycles may be able to run concurrently without violating dependencies and preventing the issue of delays that cause one cycle to run into another. Cylc was written in Python and built around a new scheduling algorithm that can manage infinite workflows of cycling tasks without a sequential cycle loop. At any point during workflow execution, only the dependence between the individual tasks matters, regardless of their particular cycle points. The information Cylc uses to control a given workflow is the task dependency. In a script file, the developers define, for each task, the parallelism settings and where data is to be stored.

Consider the Cylc workflow for a toy monthly cycling workflow in Fig. 1. In this workflow, an atmospheric model (labelled as `model` in the figure) simulates the physics from a current state to predict the future, for example, a month later. In climate research, this process is repeated in the model to simulate years into the future. Once the simulation of any month is computed, data for this month becomes available and can now be analysed. In this workflow, the task `model` is followed by tasks `postprocessing` (`post`), `forecast verification` (`ver`), and `product generation` (`prod`), all specified as a workflow in a Cylc configuration file (`flow.cylc`).

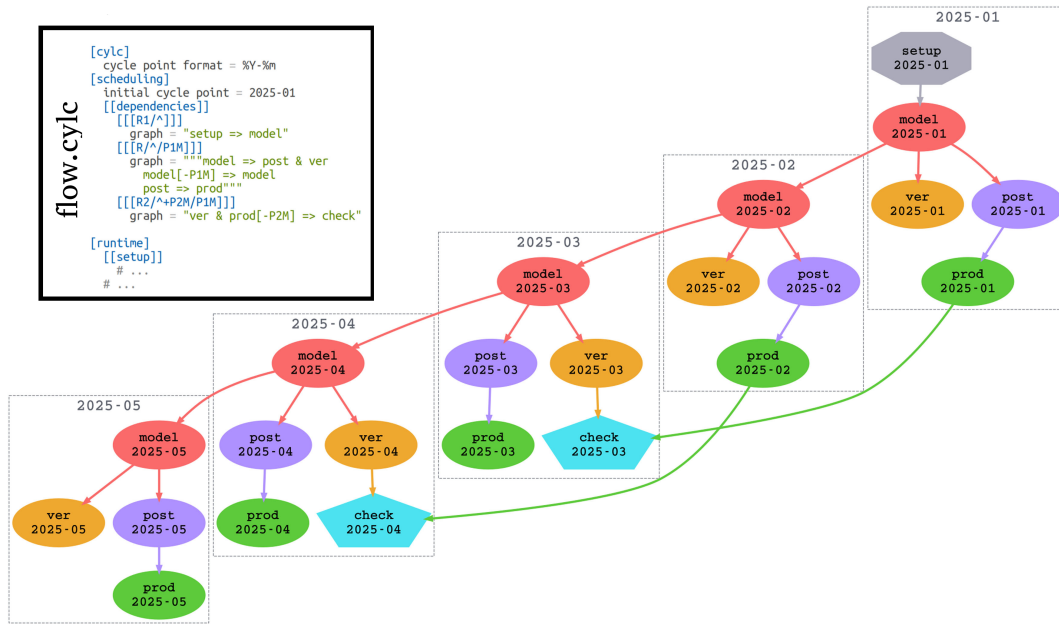


Figure 1. Example of a Cylc workflow with its configuration file [19]

1.2. Workflow Execution

While Cylc is directing the execution of workflows, several components are presented in the implementation. The software stack involved in a general workflow is depicted in Fig. 2. Next, each stage of the execution is further described.

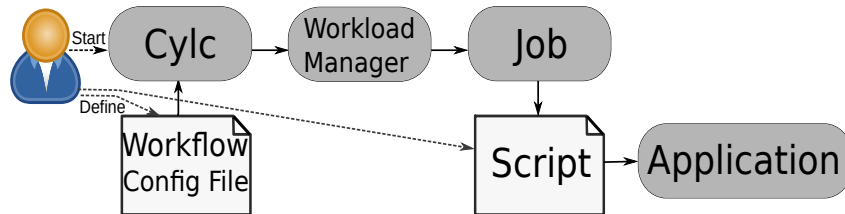


Figure 2. Software stack and stages of execution

1. **Scientist** specifies the workflow and provides a command or a script for each task. As part of the Cylc configuration, the command(s) to be run, any environment variables used by these application(s), and any workload manager directives. After that, the user enacts Cylc to start the workflow.
2. **Cylc** parses the workflow configuration file (`flow.cylc`), generates tasks dependencies, defines a schedule for the execution, and monitors the progress of the workflow. Once a task can be executed (dependencies are fulfilled), the workflow engine submits a *job script* for the workload manager with the required metadata that will run the Cylc task script.
3. **Workload Manager** such as Slurm [10] is responsible for allocating compute resources to a batch job and performing the job scheduling. The selected tool queues the job that represents the Cylc task and plans its execution, considering the scheduling policy of the data centre. Once the job is scheduled to be dispatched, i.e., resources are available, and the job priority is the highest, it is started on the supercomputer.
4. **Job** provides the environment with the resources and runs the user-provided program or script on one of the nodes allocated for it. Local variables containing information about the

environment of the batch job, e.g., the compute nodes allocated, enact the Cylc provided script on the node.

5. **Script** starts the commands sequentially (a command can be a parallel application). During the creation of the script, Cylc has included variables that describe the task in the workflow. The information is typically fed into the application(s) representing the task and defining the storage location. The script uses commands to generate filenames considering the cycle and may store data in a workflow-specific shared directory. Either these commands are set in the Cylc workflow and then injected as environment variables or directly utilised as a part of the user-provided script.
6. **Application** is executed taking the filenames set by the script.

1.3. I/O Stack of a Parallel Application

Climate applications may have complex I/O stacks, as can be seen in Fig. 3a. In this case, we assume the application uses XIOS [17], which is providing domain-specific semantics to climate and weather. It may gather data from individual fields distributed across the machine (exploiting MPI for parallelism) and then uses NetCDF [4] to store data as a file. Under the hood, NetCDF uses the HDF5 API with its file format. Internally, HDF5 uses MPI and its data types to specify the nature of data stored. Finally, data is stored on a parallel file system like Lustre which, on the server-side, stores data in a local file system on block devices such as SSDs and HDDs.

Different applications involved in a workflow may use different I/O stacks to store their outputs. Naturally, the application which uses previously generated data as its inputs must use a compatible API to read the specific data format. In Fig. 3a, for example, XIOS may perform parallel I/O via the NetCDF API, allowing subsequent processes to read data directly using NetCDF. Within the ESiWACE project², we are developing the Earth System Data Middleware (ESDM) [14] to allow applications with this kind of software stack to exploit heterogeneous storage resources in data centres. The goal of ESDM is to provide parallel I/O for parallel applications with advanced features to optimise subsequent read accesses. Implemented as a standalone API, it also provides NetCDF integration allowing its usage in existing applications. Hence, in Fig. 3a, the HDF5 layer can be replaced with ESDM.

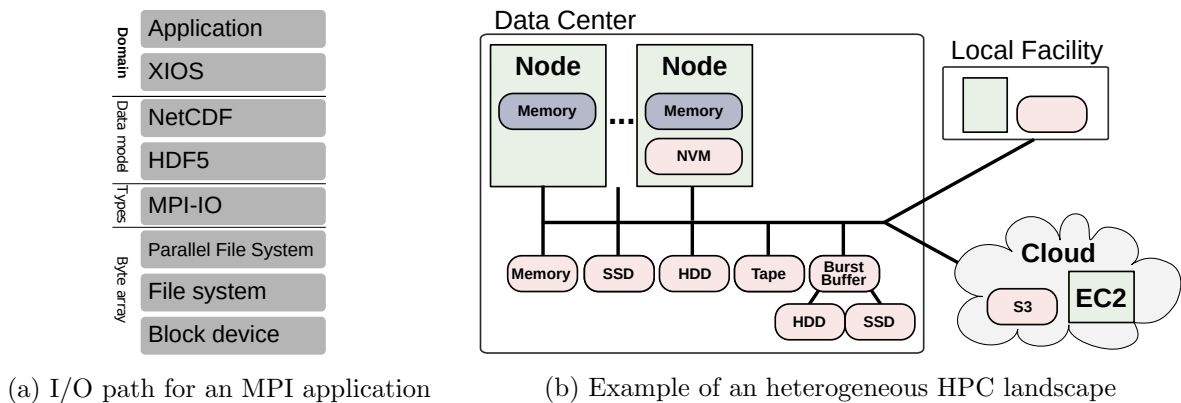


Figure 3. Typical hardware and software environment for applications

²<https://www.esiwace.eu/>

1.4. Data Centre Infrastructure

At present-day, data centres provide an infrastructure consisting of computing and storage devices with different characteristics, making them more efficient for specific tasks and satisfying the needs of different workflows. Take, for example, the supercomputer Mistral at DKRZ, that consists of 3,321 nodes³ and offers two types of compute nodes equipped with different CPUs and GPU nodes. Each node has an SSD for local storage, and DKRZ has additionally two shared Lustre file systems with different performance characteristics. Individual users and projects are mapped to one file system explicitly, and users can access it with `work` or `scratch` semantics. While data is kept on the `work` file system indefinitely, available space is limited by a quota. The `scratch` file system allows storing additional data, but data is automatically purged after some time.

Future centres are expected to have even more heterogeneity. A variety of accelerators (GPU, TPU, FPGAs), active storage, in-memory, and in-network computing technologies will provide further storage and processing capabilities. Fig. 3b shows such a system with a focus on computation and storage. Some of these technologies might be locally (specific compute nodes) or globally available. Depending on the need, the storage characteristics range from predictable low-latency (in-memory storage, NVMe) to online storage (SSD, HDD), and also cheap storage for long-term archival (tape). The tasks within any given workflow could benefit from utilising different combinations of storage and computing infrastructure.

1.5. Data Management

Usually, the scripts representing tasks define how data is placed on the available storage system. What happens in many current workflows is that they ignore the benefits of using multiple file systems concurrently and data locality between tasks to colocating them. On top of that, in the current state-of-the-art scientists optimise the available storage resources intuitively and compile the information about this decision-making process manually.

If a user knows the workflow and the system characteristics, data placement decisions can be optimised. Consider, for instance, the situation where each computing node has access to three file systems: a fast `scratch` file system on which data may reside only for a week, a slower `work` file system, and a local file system. Most current workflows utilise `work` and `scratch` systems. When a task is set to run, the corresponding dataset would be moved from `work` to `scratch`, processed, and the resulting dataset would be transferred back to `work`. If the `scratch` file system reaches its capacity, the dataset would be moved back to `work`, and the task would continue running until it is finished, which might be inefficient. In this situation, there are many straightforward opportunities to utilise data migration to optimise performance, and also other criteria (e.g., costs). However, with a multitude of file systems that differ at each data centre, such optimisations would be difficult to achieve manually by users. Policy-driven systems and burst buffers perform such optimisations automatically to some extent. However, as they lack information about the workflow, they cannot optimise workflows altogether.

³<https://www.dkrz.de/up/systems/mistral>

2. State-of-the-Art

Related work to the proposed approach can be categorised into: 1) Technology that exploits heterogeneous storage environments and supports user-directed policies and 2) Solutions for workflow processing.

Technology. Manual tiering requires the user or application to control data placement, i.e., storing data typically in the form of files on a particular storage system and, usually, moving data between storage by scripts. One limitation of such an approach is that decisions about how data are mapped and packaged into files are made by the producing application, and cannot be changed without manual intervention by a downstream application.

Burst buffer solutions provide a tiered storage system that aims to exploit a storage hierarchy. They can be integrated into hardware capabilities such as DDN’s Infinite Memory Engine (IME) [2] or simple software solutions. A policy system, e.g., deployed on a burst buffer [22], aims to simplify data movement for the user, but typically migrates objects in the coarse granularity of files. File systems and data management software such as IBM Spectrum Scale [23], HPSS [26], BeeGFS [5], and Lustre [3] (e.g., using the progressive file layouts feature) provide hierarchical storage management allowing to store data on different storage technology according to administrator-provided policies. However, the semantic information that can be used by this type of system to make decisions is limited, e.g., data location, file extension, file age, etc.

The storage community had also adjusted various higher-level software to support storage tiering on top of several storage systems. For instance, ADIOS provides in-memory staging that had been exploited by a variety of applications [24]. Hermes [12] introduces a multi-tiered I/O buffering system with pre-fetcher that provides several data placement policies. iRODS [21] is a rule-oriented data system that allows scientists to organise data into shareable collections and provides several patterns for workflows considering data locality and data migration/replication. Finally, there have also been extensions to batch schedulers to perform data staging for utilising node-local storage, for example, NORNS [18] as an extension to Slurm.

Workflows. A good overview of the flavours of Scientific Workflow Management Systems (SWfMS) and their application to data-intensive workflows is given in [15]. The article states that SWfMS should enable the parallel execution of data-intensive scientific workflows and exploit vast amounts of distributed resources. Existing solutions recognise challenges in data variety (formats of the input data), opportunities to optimise the schedule by moving code to data, specification of the data dependencies for tasks, and they even may consider the capacity of the available data storage. The execution engine Dryad [9], for example, allows transferring data between tasks via files or directly using TCP connections and attempts to schedule tasks on the same nodes or racks. In [16], an approach was presented to monitor and analyse I/O behaviour of HPC workflows. Swift/T [27], a scripting language for describing dataflow processing enabling the execution of ensembles of applications, is now openly used as a prototype platform [20]. Recent improvements aim to migrate data to a local cache allowing to exploit locality. For instance, in [6], information about locality is proposed to be stored in extended attributes.

Several early research in grid workflows and, lately, cloud workflows, use cases of interest to maximise data locality. Economic factors (including storage costs) for workflow execution are discussed in [1]. In [7], the authors discuss the role of Machine Learning (ML) for workflow execution and elaborate a general potential for resource provisionings such as optimisation of

runtime parameters, data movements, and hierarchical storage. In [25], an ML model that stages data for in-situ analysis by exploiting the access patterns is introduced.

Workflow systems can also be specifically utilised to reproduce scientific results, i.e., recompute the results. Those scalable workflow solutions typically utilise a container solution to allow execution in an arbitrary software environment. Popper [11], Snakemake [13], and Nextflow [8] provide a language to specify workflows and to execute them. Snakemake is interesting as it supports definition and inference of input and output filenames.

While various aspects of our vision have been addressed individually by related work for different domains, the high level of abstraction that we aim for and the potential it unleashes goes beyond the capabilities of existing approaches.

3. Vision for I/O-Aware Workflows

Nowadays, in order to run a job in an HPC environment efficiently, researchers have to develop profound knowledge, not only about their workflow, which is expected, but also about decisions regarding storage, communication, computing, and considerations regarding cost-efficiency of those operations. However, applied scientists should not spend much time understanding hardware characteristics and operational knowledge of running a data centre, but using their expertise to develop their work and just collect and analyse the results of their experiments.

We aim for achieving an automatic and dynamic mapping of I/O resources to workflows. Once we have an automated decision about where the job will run and how the storage will be managed, scientists can then reuse their workflow specification on any system without further modification and even without previous knowledge about the system architecture.

There are several approaches to implement the technology for the vision proposed in this work, and changes are needed in the software components to realise it. In Section 4, we will discuss a specific design for our transitional roadmap considering climate and weather workflows and tools scientists from this field already use in their routine research.

Our vision for I/O-aware workflows requires two additional pieces of information. Firstly, the user must augment the workflow description with information about I/O specifications and explicitly annotate dependencies to datasets. Secondly, details about the storage architecture must be available.

3.1. System Information

While many optimisations are possible once an abstraction is in place, the improvements we discuss here are related to the life cycle and placement of datasets into specific storage according to system performance characteristics and workflow specification. To achieve that, the system information shall comprise of all available storage systems, the system topology, and details of each available component. Simplified and complex models of the components can be included to approximate expected performance for specific I/O patterns. It is expected that the data centre (or expert user) can create such a configuration file, e.g., by using vendor-provided information or by executing benchmarks. With this information, a scheduler can make the initial data placement, transformation, and migration decisions for individual datasets during their life cycle. This separation of concerns allows us to abstract from the workflow what is essential and what a system should optimise to ensure smart usage of the available resources.

3.2. Extended Workflow Description

In general, climate and weather workflows allow specifying tasks and dependencies among them. We aim to enhance the current information with characteristics for input and output, i.e., the datasets. An example workflow with N cycles containing input datasets and (intermediate) products is illustrated in Fig. 4. Round nodes represent tasks, squared nodes represent data, and arrows indicate dependencies. In the example, Task 1 needs two datasets to perform its work, it produces Product 1, and directly communicates with Task 2. For each new cycle, the **checkpoint** from the previous cycle (Product 1) is used as input to starting the next cycle. Most of the workflow can run automatically, except for the manual quality control of the products and the final data usage of Product 3. This last step represents the typical uncertainty of data reuse, i.e., it is unclear how Product 3 will be used further. In the approach proposed in this work, each task is annotated with the required input datasets and the generated products must include metadata such as data life cycle, the value of data, and how long it should be kept. The idea here is to embrace the concept that tasks dependencies are really imposed by datasets dependencies.

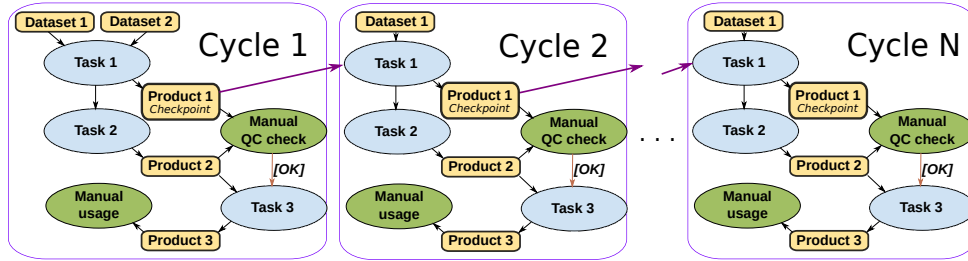


Figure 4. Example of a high-level workflow with tasks and data dependencies

3.3. Smarter I/O Scheduling

The abstraction and automation of the I/O inside a workflow allow a runtime system to improve data placement and apply data reduction on heterogeneous storage systems. Taking into consideration the architecture and workflow information, a smarter schedule can now be realised by exploiting the additional information. Value and priority can influence fault-tolerance strategies and imply the quality of service for performance and availability. Aspects like data reproducibility (can it be recomputed easily), type of the experiment (test, production), and runtime constraints for the overall and potential workflow could allow reducing costs and, hence, increase scientific output. Next, we outline two core strategies and the potential the proposed vision can bring to the improvement of current workflows. In the design proposed in this work (Section 4), we will focus on the data placement strategy.

Strategy: Data Placement Data placement encompasses all data movement-related activities such as transfer, staging, replication, space allocation and de-allocation, registering and unregistering metadata, locating and retrieving data⁴. The general idea is to host a dataset on the storage system that is most favourable in terms of performance, cost-effectiveness, and availability for the access pattern observed in the workflow. Here we are considering the optimisation of data locality, where locality is twofold, spatial and temporal, on a variety of characteristics. For optimising data placement, we introduce four approaches: data allocation, data migration, data replication and direct-coupling.

⁴<https://www.igi-global.com/dictionary/data-aware-distributed-batch-scheduling/6782>

Data Allocation is the assignment of a specific area of an available storage system to particular data. In current workflows, the user usually has a script for each task defining the filenames with a prefix that places datasets generated by the same task into a specific storage⁵. Because there is one script responsible for generating the configuration, the decision in which directory the dataset will be stored is somewhat fixed. Such configuration is done manually and with restricted information about the system architecture. It would be interesting to explore storage options for the datasets and, e.g., to have datasets from different cycles placed at different storage systems. For instance, in Fig. 4, alternating the storage location for Product 2 into two **scratch** file systems is something that would be a simple job for an I/O-aware scheduler. However, currently, it implies providing scripts for that task and all tasks depending on it with information about the different storage placement.

Data Migration is the process of transferring data from one storage system to another. Typically, it involves to delete data, but this decision can be delayed to provide read access to multiple storage systems. Data movement involves a significant overhead, both in terms of latency and energy-efficient computing, as data must be read on one storage and written to another. Hence, it needs to be considered carefully. Figure 5 introduces three possible life cycles for a specific dataset and explains how migrations can be done to improve datasets accessibility. In Fig. 5a, the dataset could be first stored on the local storage to avoid congestion on the **work** file system, then it is migrated to **work** file system where subsequent tasks of the workflow may read it multiple times. In the end, this dataset may be an intermediate product that can then be deleted. In Fig. 5b, the dataset is stored on the **scratch** file system immediately and accessed there. However, the last read access must happen before files on **scratch** are automatically removed. Alternatively, Fig. 5c presents the case where the dataset is created on **work** and it is copied to a local node. This local node allows reading accesses of subsequent tasks which might be beneficial for small random accesses. For the last two scenarios, subsequent tasks would have to be placed on the same node where previous data was stored.

Data Replication in computing involves sharing information to ensure consistency between redundant resources, such as software or hardware components, to improve reliability, fault-tolerance, or accessibility.

Data might be replicated by enabling the system to rerun parts of the workflow in case of a data loss. In addition, the system may combine the replication of data by **transforming** data into a different representation allowing to achieve better performance considering a variety of access patterns.

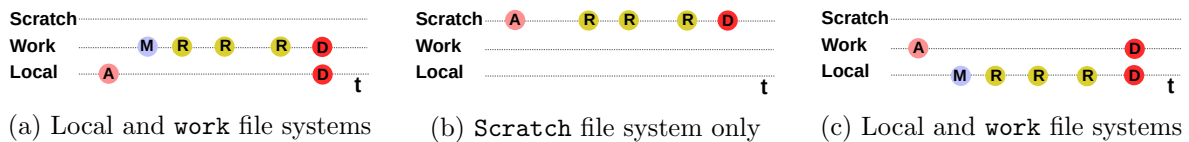


Figure 5. Alternative life cycles for mapping a dataset to storage and the operations: **A**llocation, **M**igration, **R**eadings, and **D**eleting

⁵Complicated scripts would have allowed changing the storage type depending on the cycle. Still, it is a significant burden to the user.

Direct-Coupling replaces I/O by communicating data between subsequent steps of a workflow directly without storing intermediate data products on persistent storage. As an example, in Fig. 4, the outcome of Task 1 may be used directly by Task 2. Data may also be kept in memory and cached to achieve a certain level of independence between producer and consumer.

Strategy: Data Reduction Data reduction decreases the amount of data stored. We discuss here two potential optimisations: data compression and data recomputation.

Data Compression is the process of encoding information using fewer bits than the original representation. Knowing the characteristics of data production and usage allows scientists to annotate the required precision of data in those workflows. The storage system can exploit such information by reducing the precision of data and automatically picking an appropriate compression algorithm.

Data Recomputation Climate/weather scientists are trading recomputation with space usage manually. By knowing how to rerun the workflow behind the data creation, a smarter storage system can automatically trade data availability for potential recomputation opportunities to optimise the cost-efficiency of the system. Intermediate states could be rerun by utilising virtualisation and container technologies. Consider Fig. 4 again and that, at every K cycles of the workflow, the generated Product 3 (from Cycle 1 to Cycle K) are used in a validation task, called here **check**. From the workflow, we know that P_3C_1 ⁶ will be used to construct P_3C_2 and then **check**. This dataset would probably be stored somewhere, and it will not be used until the workflow reaches the K -th cycle. One alternative is to delete it after it was first used and then recompute it when time is right. The cost of doing that is storing **checkpoint** and then use it to reconstruct P_3C_1 . If, for instance, P_3C_1 is a large dataset, **checkpoint** is small, and computing time is short, it is easy to see that deleting and recomputing it may improve the costs of running the workflow. That is just an example, and, currently, scientists perform those optimisations manually.

3.4. Benefit

The benefits of the proposed vision are:

Abstraction Providing the abstraction that enables a separation of concerns. Once the I/O characteristics of a workflow are defined, the user does not have to know the architecture of the target system on which the workflow will run. Thus, this level of abstraction can remove the specialist from the decision-making process of individual workflows.

Optimisation The workflow will be optimised specifically for the available system infrastructure and information about data. In particular, by exposing the heterogeneous architecture, potential runtime characteristics can be considered. By using information about the value of data, policies for data management (storage resilience, recomputation, replication, etc.) can be decided.

Performance-portability With both abstraction and optimisation, the user can specify the I/O requirements only once for the tasks of a specific workflow, and the I/O-aware workflow

⁶The P_iC_j notation represents the Product i generated in the Cycle j .

can now run with optimised data storage on any system without user intervention. Even more, if the system characteristics change, e.g., it gets upgraded, an additional storage tier becomes available, or if storage degrades, the I/O-aware workflow could automatically adapt and make use of this new environment.

4. Design

This section describes our first approach to incrementally extend workflows for climate and weather that realises parts of our vision. While individual components such as ESDM and Cylc exist, we have not implemented the described scheduler, yet. To automatically make scheduling decisions, the software stack needs to:

1. deliver information about dataset life cycle together with the workflow, and
2. adapt the resulting workflow, individual scripts, and application executions to consider the potential for data placement strategies.

4.1. System Information

The system information of the design is realized using already available capabilities in the ESDM middleware. We assume ESDM is used as the I/O middleware in the parallel application (with NetCDF or directly) and orchestrates the I/O according to a simplified ESDM configuration file (`esdm.conf`). This file contains information about the available technology in the data centre, its I/O characteristics, and will be used to make decisions about how to prioritise I/O targets. In the example presented in Fig. 6, we have three storage targets: two global accessible file systems (`lustre01` and `lustre02`), and one local file system in `/tmp` that can be accessed via the POSIX backend. Each of them comes with a lightweight performance model and the maximum size of data fragments. The metadata section (Line 24) utilises here a POSIX interface to store the information about ESDM objects. Internally, ESDM creates so-called containers and dataset objects to manage data fragments.

ESDM manages a pool of threads that should be created per compute node to achieve better performance and delegates the assignment of optimal block sizes to the storage backend. Since ESDM supports several (non-POSIX) storage backends, an application can utilise all available storage systems without any modifications to the code. The configuration file is inquired by the application utilising ESDM and steers the distribution of data during I/O. To elucidate the system's behaviour, ESDM distributes a single dataset across multiple storage devices depending on their characteristics. While the current system information and performance model are based on latency and throughput only, ESDM shows that automatic decision making can be made on behalf of the user.

4.2. Extended Workflow Description

The user now has to provide information about the datasets required as input and the generated output for each workflow task in a file called I/O-workflow configuration file (`io.cylc`). An example of an `io.cylc` file is shown in Fig. 7. In this file, information about Task 1 is given as an example, and we expect the extra information about all tasks in the same file. This file could define a cycle flexibly to be a month or a year according to the file `flow.cylc`. The notation

```

1 "backends": [
2   {"type": "POSIX", "id": "work1", "target": "/work/lustre01/projectX/",
3     "performance-model" : {"latency" : 0.00001, "throughput" : 500000.0},
4     "max-threads-per-node" : 8,
5     "max-fragment-size" : 104857600,
6     "max-global-threads" : 200,
7     "accessibility" : "global"
8   },
9   {"type": "POSIX", "id": "work2", "target": "/work/lustre02/projectX/",
10     "performance-model" : {"latency" : 0.00001, "throughput" : 200000.0},
11     "max-threads-per-node" : 8,
12     "max-fragment-size" : 104857600,
13     "max-global-threads" : 200,
14     "accessibility" : "global"
15   },
16   {"type": "POSIX", "id": "tmp", "target": "/tmp/esdm/",
17     "performance-model" : {"latency" : 0.00001, "throughput" : 200.0},
18     "max-threads-per-node" : 0,
19     "max-fragment-size" : 10485760,
20     "max-global-threads" : 0,
21     "accessibility" : "local"
22   }
23 ],
24 "metadata": {"type": "POSIX",
25   "id" : "md",
26   "target" : "./metadata",
27   "accessibility" : "global"
28 }

```

Figure 6. Example of an ESDM configuration file (`esdm.conf`)

is similar to the specification of Cylc workflows using a nested INI format and, ultimately, files `io.cylc` and `flow.cylc` can be merged.

For each task, inputs and outputs are defined. In the **input** section, each entry specifies the virtual name that is used by ESDM as a filename inside NetCDF. Line 5, for example, it defines that the filename `topography` is mapped to a specific input file and that this dataset does not depend on any previous step of the workflow. The next line specifies that the input filename `checkpoint` should be mapped to the output of Task 1 `checkpoint` dataset from the previous cycle (e.g., the checkpoint generated after completing the last year's production). For the initial cycle, the checkpoint file will be empty, and the application will load `init` data. In the **output** section, the datasets are annotated with their characteristics more precisely. For each variable, a pattern defining how frequently data is output according to the workflow must be provided. Most data is input and output in the periodicity of the cycle. Still, we can have variables with different patterns, such as `varA`, which is output per day regardless of the cycle.

Next, we formally define the expected annotations in all the fields envisioned in the I/O-workflow configuration file:

Name A primary name for the field/data generated. It is extended by a pattern defined in a variable (Lines: 11, 19, 26).

Pattern The frequency of data output (Lines: 12, 20).

Lifetime How long data must be retained on storage (if at all) (Lines: 13, 21).

Type The class type of data, i.e., checkpoint, diagnostics, temporary (Lines: 14, 22, 27).

Datatype The data type of data (Lines: 15, 23, 28).

Size An estimate of data size⁷ (Lines: 16, 29).

⁷This field can be inferred if dimension and data type are provided.

```

1 [Task 1]
2
3 [[inputs]]
4
5     topography = "/pool/input/app/config/topography.dat"
6     checkpoint = "[Task 1].checkpoint$(CYCLE - 1)"
7     init = "/pool/input/app/config/init.dat"
8
9 [[outputs]]
10
11     [[varA]] # This is the name of the variable
12     pattern = 1 day
13     lifetime = 5 years
14     type = product
15     datatype = float
16     size = 100 GB
17     precision.absolute_tolerance = 0.1
18
19     [[checkpoint]]
20     pattern = $(CYCLE)
21     lifetime = 7 days
22     type = checkpoint
23     datatype = float
24     dimension = (100,100,100,50)
25
26     [[log]]
27     type = logfile
28     datatype = text
29     size = small

```

Figure 7. External Cylc I/O-workflow configuration file (`io.cylc`)

Dimension The data dimension (Line: 24).

Accuracy Characteristics quantifying the required level of data precision (Line: 17).

Note that the user may not be able to provide all required information which can be handled by assuming a default safe behaviour. For instance, in the case of missing data precision, data should be retained in the original form. Knowing the dimension or size a priori might be difficult for scientists, e.g., the log file size is unclear. In this case, the user may insert relevant information like small or big, indicating that any information is better than no information at all. In future, we will explore ways to infer the output volume from the input automatically. For instance, by running the workflow without I/O specification and monitoring I/O accesses for one cycle, we can propose an I/O description to the user to simplify the specification and generate an experimental I/O configuration file.

4.3. Smarter I/O Scheduling

From the list of opportunities, we realise data placement and migration in a heterogeneous (multi-storage) environment. These goals will be achieved via the proposed I/O-aware scheduler, called here EIOS (ESDM I/O Scheduler). EIOS will make the schedule considering Cylc workflow and ESDM provided system characteristics. We are working together with Cylc Team in developing how EIOS interfaces with Cylc. While Cylc schedules the workflow, EIOS can provide hints about colocating tasks which generate the opportunity for keeping data in local storage. Our design imposes only minor changes to Cylc as normal functionalities cover the core requirements:

The ability to dynamically set the job (Slurm) directives for a task

This will be achieved by calling an external command (run on the Cylc scheduler host) which adds additional directives to be used by the job. This command, provided by EIOS, will determine attributes of previous tasks through simple SQL queries to the Cylc database. We plan on using the Cylc broadcast functionality to change the instructions used by a task by running an external program before any task where we may want to alter the directives.

The ability to dynamically set storage locations

This will be achieved by defining environment variables in the job script which are set to the output of another external command (run on the job host). This command, also provided by EIOS, will have access to all the standard Cylc environment variables with details about the current task.

We plan on utilising DDN's IME API to pin data in IME and trigger migrations between IME and a storage backend explicitly. Decisions about data locality will not be made for a whole (and potentially big) workflow. Instead, the system will make decisions by looking ahead to several steps of the workflow, allowing reacting to the observed dynamics of the execution. Ultimately, when a user-script runs, the information about the intended I/O schedule is communicated from EIOS through a modified filename, which is then used by the ESDM-aware application to determine the data placement.

4.4. Modified Workflow Execution

The steps to execute a workflow enriched with I/O information and perform smarter scheduling are depicted in Fig. 8. Components of EIOS are involved in different steps of the workflow and the I/O path. The suggested alterations can be seen in boxes pointed by red arrows, and the remaining components are the current state-of-the-art for workflows in climate and weather from Fig. 2. In the following, we describe the modifications we propose in this vision paper for each component involved in the software stack.

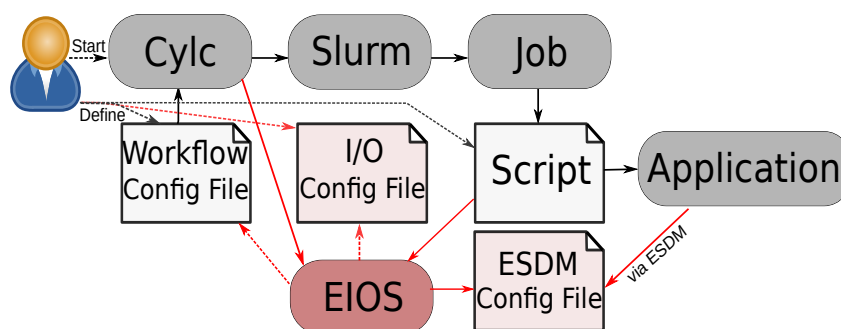


Figure 8. Software stack and stages of execution with the I/O-aware scheduler (EIOS). The red arrows and boxes indicate additions to the workflow compared to Fig. 2

1. **Scientist** The user now has to provide an additional file that covers the I/O information for each task and slight changes have to be made to the current scripts.
2. **Cylc** EIOS is invoked by Cylc to identify potential optimisations in the schedule before generating the Slurm script.

3. **EIOS** The ESDM I/O Scheduler reads the information about the workflow (`flow.cylc` and `io.cylc` configuration files) and acts depending on the stage of the execution. EIOS consists of several subcomponents:
 - The high-level scheduler that interfaces with Cylc.
 - A tool to generate pseudo filenames used by the ESDM-aware applications.
 - A data management service (not shown in the figure) that migrate and purge data at the end of the life cycle.

EIOS components use knowledge about the system by parsing the `esdm.conf` file. EIOS may decide that subsequent jobs shall be placed on the same node, reorder the execution of some jobs, and provide information for conducting data migration.
4. **Slurm** Cylc may now have added an optimisation identified by EIOS which is promptly handled by a modified Slurm. Also, if migrations have to be performed, Slurm will administer them according to the specification in the job script.
5. **Job** A job might run on the same node as a previous job to utilise local storage.
6. **Script** Filenames are now generated by a replacement command that calls EIOS to create a pseudo filename. This filename will encode additional information for ESDM about how to prioritise data placement according to data access.
7. **Application** The application may either use XIOS, NetCDF with ESDM support or ESDM directly to access datasets. ESDM loads the file `esdm.conf` that contains the information about the available storage backends and their characteristics. ESDM extracts the long-term schedule information from the generated pseudo filenames and employs them during the I/O scheduling to optimise the storage considering data locality among tasks. Basically, ESDM can now change the priorities for data placement in the different storage locations that would typically be encoded in Cylc's configuration file.

5. Potential Benefit

In this section, we discuss the potential performance benefit that our vision for I/O-aware workflows may have considering DKRZ Mistral supercomputer, ESDM current version and a hypothetical workload related to the workflow in Fig. 1. In our scenario, we compare the usage of the node-local file system⁸ with a globally shared Lustre file system to store intermediate data. We focus on the model execution and subsequent verification and postprocessing steps. Firstly, checkpoints of a long model execution chain could be stored locally and restart from there. When the subsequent jobs require the whole data to generate a product, they can be run on the same nodes.

Figure 9 shows the read/write performance when using ESDM to store a time series of 10 steps of a variable with $200\text{ k} \times 200\text{ k}$ dimension (about 1 km global resolution of the model, equivalent to 3 TB of data in total). Note that, while we only consider the volume data for one variable with one level and ten timesteps, this value could be multiplied by a sensible number of levels and timesteps of the model. Data is stored on either Lustre02 or both Lustre file systems – ESDM splits data of a single variable internally and distributes them across the file systems. While the mapping is not yet optimal, the figure shows that the write performance benefits from this approach. In our observations, performance will not improve beyond 500 nodes, which

⁸We assume the availability and fault-tolerance of the nodes is non-uncritical for the particular workload – typically nodes can be repaired and returned to the pool within days.

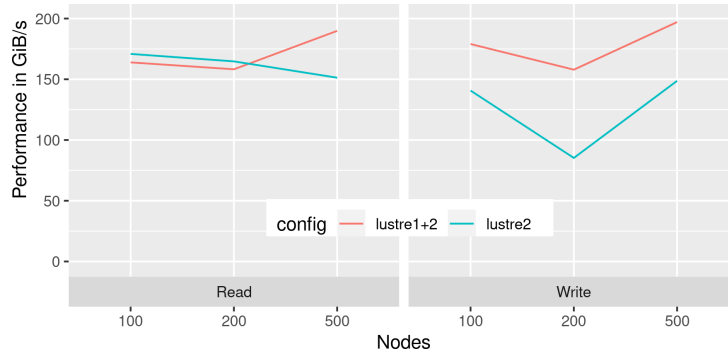


Figure 9. Lustre performance for 100, 200, and 500 nodes

might be due to the fact that each Lustre file system has 128 Object Storage Targets (256 in total).

Another improvement can be achieved by using local storage. Each local SSD of Mistral is a Micron M600 MTFD⁹ (256 GB), which has a nominal sequential read/write performance slightly above 500 MB/s. Hence, with 500 nodes, we could achieve 250 GB/s, which surpasses the Lustre performance observed in Fig. 9. Even more, our 3 TB of data would be about 6 GB per node, which could be cached in memory and overlap with the computation phases. An additional benefit of using local storage is that the interference with I/O activities of other jobs would be minimised. Actually, for all reasonable sizes of the experimental data with 500+ nodes, the observed performance of node-local storage would be higher, and thus, improving the workflow execution time. Since DKRZ has more than 3,000 nodes, using the local SSD would sum up to 1.5 TB/s speeding up the IO phase by 7x. For model runs with 1 km resolution, such configurations would be reasonable. It might also be suitable to couple the model with a parallel analysis process directly using an in-memory file system such as `tmpfs`. In this case, the performance per node can be assumed to be 3 GiB/s, making it a viable option for smaller runs.

Users can always use the node-local storage manually and create specific run-scripts to reproduce similar behaviours utilising the local storage. However, this would be tedious and error-prone. The purpose of our proposal is to establish an abstraction layer to allow for semi-automatic decision making and reduce, or even remove manual intervention.

Conclusions

In the domain of climate and weather, organising data placement on storage tiers is performed by the users or via policies, often leading to suboptimal decisions. Additionally, manual optimisation and hard-coding of storage locations are non-portable and an error-prone task. We believe users must be able to express their workflows abstractly. By increasing the abstraction level for scientists, not only tedious manual optimisation could be automatised, but also strategies for data placement and data reduction can be harnessed. With knowledge about the data pattern, the runtime system could generate optimised execution plans and monitor their execution. In this work, we describe the overall vision and a specific design for the software stack in the domain of climate and weather that we work on in the ESiWACE project. The proposed changes increase the opportunity for smarter scheduling of storage in heterogeneous storage environments by considering the characteristics of data and system architecture in the workflow.

⁹<https://www.anandtech.com/show/8528/micron-m600-128gb-256gb-1tb-ssd-review-nda-placeholder>

Acknowledgements

This project is funded by the European Union's Horizon 2020 research and innovation programme under grant agreement No. 823988. We thank our collaborators Bryan Lawrence, Glenn Greed, David Matthews, and Hua Huang for their input to this paper, and the NGI initiative for contributions to the vision.

This paper is distributed under the terms of the Creative Commons Attribution-Non Commercial 3.0 License which permits non-commercial use, reproduction and distribution of the work without further permission provided the original work is properly cited.

References

1. Alkhanak, E.N., Lee, S.P., Rezaei, R., Parizi, R.M.: Cost optimization approaches for scientific workflow scheduling in cloud and grid computing: a review, classifications, and open issues. *Journal of Systems and Software* 113, 1–26 (2016), DOI: 10.1016/j.jss.2015.11.023
2. Betke, E., Kunkel, J.: Benefit of DDN's IME-Fuse and IME-Lustre file systems for I/O intensive HPC applications. In: Yokota, R., Weiland, M., Shalf, J., Alam, S. (eds.) *High Performance Computing: ISC High Performance 2018 International Workshops, Frankfurt/Main, Germany, 28 June, 2018, Revised Selected Papers. Lecture Notes in Computer Science*, vol. 11203, pp. 131–144. ISC Team, Springer (2019), DOI: 10.1007/978-3-030-02465-9_9
3. Braam, P.: The Lustre storage architecture. CoRR abs/1903.01955 (2019), <http://arxiv.org/abs/1903.01955>
4. Center, U.P.: Network Common Data Form (NetCDF), DOI: 10.5065/D6H70CW6
5. Chowdhury, F., Zhu, Y., Heer, T., Paredes, S., Moody, A.T., Goldstone, R., Mohror, K.M., Yu, W.: The parallel I/O architecture of the high-performance storage system (HPSS). In: *Proceedings of the 48th International Conference on Parallel Processing, August 2019, Kyoto, Japan*. pp. 1–10 (2019), DOI: 10.1145/3337821.3337902
6. Dai, D., Ross, R., Khaldi, D., Yan, Y., Dorier, M., Tavakoli, N., Chen, Y.: A cross-layer solution in scientific workflow system for tackling data movement challenge. CoRR abs/1805.061675 (2018), <https://arxiv.org/abs/1805.06167>
7. Deelman, E., Mandal, A., Jiang, M., Sakellariou, R.: The role of machine learning in scientific workflows. *The International Journal of High Performance Computing Applications* 33(6), 1128–1139 (2019), DOI: 10.1177/1094342019852127
8. Di Tommaso, P., Chatzou, M., Floden, E.W., Barja, P., Palumbo, E., Notredame, C.: Nextflow enables reproducible computational workflows. *Nature Biotechnology* 35, 316–319 (2017), DOI: 10.1038/nbt.3820
9. Isard, M., Budiu, M., Yu, Y., Birrell, A., Fetterly, D.: Dryad: Distributed data-parallel programs from sequential building blocks. In: *Proceedings of the 2nd ACM SIGOPS/EuroSys European Conference on Computer Systems 2007, March 2007, Lisbon, Portugal*. p. 59–72. Association for Computing Machinery, New York, NY, USA (2007), DOI: 10.1145/1272996.1273005
10. Jette, M.A., Yoo, A.B., Grondona, M.: SLURM: Simple Linux Utility for Resource Management. In: *Proceedings of Job Scheduling Strategies for Parallel Processing, 24 June, Seattle, WA, USA. Lecture Notes in Computer Science*, vol. 2862, pp. 44–60. Springer, Berlin, Heidelberg (2002), DOI: 10.1007/10968987_3

11. Jimenez, I., Sevilla, M., Watkins, N., Maltzahn, C., Lofstead, J., Mohror, K., Arpaci-Dusseau, A., Arpaci-Dusseau, R.: The popper convention: making reproducible systems evaluation practical. In: 2017 IEEE International Parallel and Distributed Processing Symposium Workshops, 29 May-2 June 2017, Lake Buena Vista, FL, USA. pp. 1561–1570. IEEE (2017), DOI: 10.1109/IPDPSW.2017.157
12. Kougkas, A., Devarajan, H., Sun, X.H.: I/O acceleration via multi-tiered data buffering and prefetching. *Journal of Computer Science and Technology* 35(1), 92–120 (2020), DOI: 10.1007/s11390-020-9781-1
13. Köster, J., Rahmann, S.: Snakemake: a scalable bioinformatics workflow engine. *Bioinformatics* 28(19), 2520–2522 (2012), DOI: 10.1093/bioinformatics/bts480
14. Lawrence, B.N., Kunkel, J.M., Churchill, J., Massey, N., Kershaw, P., Pritchard, M.: Beating data bottlenecks in weather and climate science. In: *Extreme Data Workshop – Forschungszentrum Jülich, Proceedings, IAS series. vol. 40*, pp. 31–36 (2018)
15. Liu, J., Pacitti, E., Valduriez, P., Mattoso, M.: A survey of data-intensive scientific workflow management. *Journal of Grid Computing* 13(4), 457–493 (2015), DOI: 10.1007/s10723-015-9329-8
16. Lüttgau, J., Snyder, S., Carns, P., Wozniak, J.M., Kunkel, J., Ludwig, T.: Toward understanding I/O behavior in HPC workflows. In: *IEEE/ACM 3rd International Workshop on Parallel Data Storage & Data Intensive Scalable Computing Systems*, 12 Nov. 2018, Dallas, Texas. pp. 64–75. IEEE Computer Society, Washington, DC, USA (2019), DOI: 10.1109/PDSW-DISCS.2018.00012
17. Meurdesoif, Y., Caubel, A., Lacroix, R., D’erouillat, J., Nguyen, M.H.: XIOS Tutorial (2016), <http://forge.ipsl.jussieu.fr/ioserver/raw-attachment/wiki/WikiStart/XIOS-tutorial.pdf>
18. Miranda, A., Jackson, A., Tocci, T., Panourgias, I., Nou, R.: NORNS: extending Slurm to support data-driven workflows through asynchronous data staging. In: *2019 IEEE International Conference on Cluster Computing*, 23-26 Sept. 2019, Albuquerque, NM, USA. pp. 1–12. IEEE (2019), DOI: 10.1109/CLUSTER.2019.8891014
19. Oliver, H., Shin, M., Matthews, D., Sanders, O., Bartholomew, S., Clark, A., Fitzpatrick, B., van Haren, R., Hut, R., Drost, N.: Workflow automation for cycling systems: the Cylc workflow engine. *Computing in Science Engineering* 21(4), 7–21 (2019), DOI: 10.1109/MCSE.2019.2906593
20. Ozik, J., Collier, N.T., Wozniak, J.M., Spagnuolo, C.: From desktop to large-scale model exploration with Swift/T. In: *2016 Winter Simulation Conference*, 11-14 Dec. 2016, Washington, DC, USA. pp. 206–220. IEEE (2016), DOI: 10.1109/WSC.2016.7822090
21. Rajasekar, A., Moore, R., Hou, C.y., Lee, C.A., et al.: iRODS primer: integrated rule-oriented data system. *Synthesis Lectures on Information Concepts, Retrieval, and Services* 2(1), 1–143 (2010), DOI: 10.2200/S00233ED1V01Y200912ICR012
22. Romanus, M., Ross, R.B., Parashar, M.: Challenges and considerations for utilizing burst buffers in high-performance computing. *CoRR abs/1509.05492* (2015), <http://arxiv.org/abs/1509.05492>
23. Schmuck, F., Haskin, R.: Gpfs: A shared-disk file system for large computing clusters. In: *Proceedings of the 1st USENIX Conference on File and Storage Technologies*, Monterey, CA. pp. 231–244. USENIX Association, USA (2002), DOI: 10.5555/1083323.1083349
24. Slawinska, M., Clark, M., Wolf, M., Bode, T., Zou, H., Laguna, P., Logan, J., Kinsey, M., Klasky, S.: A Maya use case: adaptable scientific workflows with ADIOS for general relativistic astrophysics. In: *Proceedings of the Conference on Extreme Science and Engineering Discovery Environment: Gateway to Discovery*, July 2013, San Diego, California, USA. pp. 1–8. Association for Computing Machinery, New York, NY, USA (2013), DOI: 10.1145/2484762.2484795

25. Subedi, P., Davis, P.E., Parashar, M.: Leveraging machine learning for anticipatory data delivery in extreme scale in-situ workflows. In: 2019 IEEE International Conference on Cluster Computing, 23-26 Sept. 2019, Albuquerque, NM, USA. pp. 1–11. IEEE (2019), DOI: 10.1109/CLUSTER.2019.8891003
26. Watson, R.W., Coyne, R.A.: The parallel I/O architecture of the high-performance storage system, 11-14 Sept. 1995, Monterey, CA, USA. In: Proceedings of IEEE 14th Symposium on Mass Storage Systems. pp. 27–44. IEEE (1995), DOI: 10.1109/MASS.1995.528214
27. Wozniak, J.M., Armstrong, T.G., Wilde, M., Katz, D.S., Lusk, E., Foster, I.T.: Swift/T: Large-Scale Application Composition via Distributed-Memory Dataflow Processing. In: 2013 13th IEEE/ACM International Symposium on Cluster, Cloud, and Grid Computing, 13-16 May 2013, Delft, Netherlands. pp. 95–102. IEEE (2013), DOI: 10.1109/CCGrid.2013.99