# Governance-Centric Paradigm: Overcoming the Information Gap between Users and Systems by Enforcing Data Management Plans on HPC-Systems

Hendrik Nolte
*GWDG*
Göttingen, Germany
hendrik.nolte@gwdg.de

Julian Kunkel
*University of Göttingen*
Göttingen, Germany
julian.kunkel@gwdg.de

***Abstract*—Along with the increase in available compute power of High-Performance Computing (HPC) systems and the success of novel data-driven methods, the amount of data processed and the user groups increase as well. This gave rise to two big challenges: The traditional interaction scheme of users with modern HPC systems becomes more and more unsuited to deal with large data sets and many independent tasks working on these data sets. This highly manual way can quickly lead to unreproducible results and data loss due to missing backups since it is stored fragmented on multiple storage tiers. Similarly, domain-specific data management systems have been established to ease the burden of data and process management of particularly inexperienced users. These systems, however, only offer a very rigid, and tool-specific interaction scheme. This resulted in a gap between these two user groups, which even hinders large-scale cooperations across different domains. In this paper, we introduce the Governance-Centric interaction paradigm, a novel, and holistic concept which allows us to enforce data management plans to bridge this gap.**

***Index Terms*—data management, high-performance computing, provenance, reproducibility, IO performance.**

## I. INTRODUCTION

Data-driven methods gained a lot of momentum in recent years and their success lead to adoptions in a wide variety of scientific domains. Also, many sciences are data-intense such as climate/weather. These data-driven projects have a few things in common. First, they require large data sets to be able to derive results with good statistics. Second, these large data sets often have large storage requirements due to their size. Third, these data sets often consist of millions of small files which are typically organized in storage within a few flat namespaces. However, these methods are not only data-intensive, but processing all of these data sets in a reasonable amount of time requires large compute resources. Therefore, researchers have started to utilize High-Performance Computing (HPC) clusters to serve those tasks.

There are various challenges when handling and processing this data. **1) Performance**: these iterative procedures on these small files lead to heavy loads on the storage system, which can overload, particularly the metadata servers, and can lead to large performance degradation due to storage bottlenecks. **2) Data management**: fulfilling the FAIR principles [1], i.e., making data findable, accessible, interoperable, and reusable is challenging. For instance in order to make data findable a naming scheme is mandatory - coming up with a naming scheme for files/objects created and then actually following it. Sharing data with other researchers often comes as an afterthought. It is a reasonable assumption, that most projects do neither strictly follow the FAIR principles nor their Data Management Plan (DMP) if there was one defined at the beginning of a project. It can be expected that this issue will only be exacerbated by the increasing complexity and heterogeneity of the employed storage systems in the compute continuum. **3) Integration of compute and data handling**: Computing on the HPC system feels a bit archaic. Users have to manually define many system settings, for instance, file names define what storage to use. Meanwhile, the complexity of the tiered storage systems in modern HPC systems has drastically increased. There exists no way to define and enforce data governance, which is homogeneously applicable across all of the disparate storage tiers. **4) Reproducibility**: Being able to understand the lineage of data and how to reproduce certain outputs is important for trust in the scientific results. However, as execution on HPC systems are usually scripts that are invoked manually, on binaries created specifically for the given supercomputer, it is tricky to reproduce results.

We do not list usability as an independent key challenge on its own explicitly, as it is primarily a function of data management and integration.

One promising solution for 2) would be to use new or established Data Management Systems (DMS) in these data-intensive projects. These systems could provide a unified namespace across a tiered and distributed storage architecture by offering a single point for data copies to reside. However, this requires tight integration of the tools in HPC systems. Additionally, since there will always be a gap between a remote DMS and a HPC system, ensuring reliable information within the DMS which originates from a HPC system is an unsolved problem. In this article, we systematically discuss an overarching next-generation concept to integrate DMS into data-intensive HPC workflows and create a user-friendly unified infrastructure for compute and storage that we believe

HPC should be. This includes the following contributions:

- the current user interaction paradigms with HPC systems are discussed and classified
- the involved components are discussed
- the prevailing gap between different user groups is identified using a layer model
- the novel governance-centric paradigm is proposed

The remainder of this paper is structured as follows: in Section II, the related work is discussed, leading to the discussion of the prevailing interaction paradigms in Section III. Based on this, the novel governance-centric paradigm is presented in Section IV, which is followed by the conclusion in Section V.

## II. Context and Related Work

In the following, we describe the state-of-the-art in our four challenges.

### A. Performance

Usually, HPC clusters provide at least two parallel file systems, providing file access via Portable Operating System Interface I/O (POSIX-IO) semantics, the relaxed Message Passing Interface I/O (MPI-IO) [2] semantics or the close-to-open semantics used by the Network File System (NFS), to name just a few. All of these semantics require dedicated metadata servers to empower parallel file systems, like Lustre [3], or the General Parallel File System (GPFS) [4]. These metadata servers handle all metadata operations using special data structures called inodes to handle these metadata operations. If an inode represents a folder on such a filesystem, it contains a list of all inodes located in this folder. Depending on the actual operation, which should be done, it might be necessary to also read additional information from each inode within a folder, for example, the file permissions, or the ownership. The cost for these metadata operations scales linearly with the number of files stored within a single folder. However, if the list of inodes stored in a single inode becomes too long, indirect inodes have to be used. This behavior can be triggered if those inode lists are inlined within a small data block within the inode itself. This is typically done to avoid lookups on the storage servers holding the actual data of a file, which would otherwise increase the latency of such a metadata operation drastically. These indirect inodes, potentially even consisting of multiple layers, lead to an even worse performance degradation. Therefore, having too many files within a single folder has a huge performance penalty. However, this can often be observed in machine learning projects, e.g., if there a tens of thousands of small images in a single folder whose name encodes the particular target, e.g., a folder called *cats* containing many small images of cats.

Although there is a varying amount of overhead necessary in the different semantics and filesystems, they all share the problem of bottlenecking when exposed to this described small file IO. Current mitigation strategies consist of either providing a multi-tier storage system, where each tier is optimized to handle certain workloads, or meeting a specific cost-to-capacity ratio. This option leads to increased complexity and requires the users to manually move and stage data to the correct tiers to achieve optimal performance while ensuring that cold data is not piling up on fast and expensive storage, which is not backed up. In addition, novel storage concepts, like object storages, are being integrated into HPC cluster, which supports flat namespaces by design. These are already common in cloud environments, with prominent standards like Amazon's S3, or Openstack Swift. However, their REST-based interfaces entail additional overhead, both, on the communication layer, and also on the application layer, since the file handling drastically differs from the well-established POSIX-IO compatible file systems.

### B. Data Management

There are already some established tools that try to abstract and simplify the interaction with complex and heterogeneous HPC systems. One of these tools is *VIKING* [5] which is used specifically for molecular dynamics simulations and provides a user-friendly web interface to run *NAMD* [6] or *Gromacs* [7] among others.

Similarly, *XNAT* [8] is a DMS specifically built for neuroimaging data. It allows the organization of data within a hierarchical structure consisting of projects, subjects, and experiments. Analysis can be done and solely controlled from within the web interface using *Docker* container on a dedicated *Docker-Swarm* or *Kubernetes* cluster.

However, these tools only provide a very restrictive compute model or require a lot of manual steps by the users to allow for larger flexibility.

### C. Integration of Compute and Data Handling

In order to integrate HPC systems into XNAT, DAX [9] was developed. It supports the execution of preconfigured tasks, called *Spiders* on a batch system, but does not support direct access to the data or the tasks by the users on the HPC system.

In cloud environments, the integration of compute and data is well established and is commonly implemented in any Infrastructure-as-a-Service offering. The entire approach can even be found on Hadoop systems, where tasks and data were transparently integrated either by batch jobs accessing data via the Hadoop distributed file system [10] or by interactive tasks using the YARN [11] resource manager.

Today, similar approaches can be seen with Jupyter-Hub deployments in HPC centers [12], which, however, do not lift the burden of managing tiered storage systems of the user.

Therefore, data and compute are currently considered separate parameters, a user has to individually and manually manage and optimize. Particularly, the integration into a user's overarching experiment is lacking, since no globally defined data governance can be homogeneously enforced across all storage tiers.

### D. Reproducibility

Since mostly data-parallelism is assumed in these data-intensive workloads, the question of reproducibility of HPC jobs is reduced to the problem of provenance auditing while the deterministic execution of a HPC job is completely neglected. There is related work for provenance auditing on HPC systems. Two often used approaches are either monitoring system calls like *PASS* [13] to create audit trails. Following a similar approach, *LPS* [14] has drastically reduced the runtime overhead, but is not completely transparent due to the use of a dedicated *Library Wrapper*. *ReproZip* [15] also continues the idea of audit trails of system calls to automatically build packages to re-run an experiment.

A different way for lineage recording is provided by *Data Pallets* [16]. Here, all processes run within containers where all write access to the storage devices is intercepted and transparently redirected to data containers. Hereby, all data containers are automatically annotated with reliable provenance recordings.

Although provenance tools for HPC systems have evolved, they currently lack integration of containers, and awareness of DMS, i.e. if a problem data management is done, input data can be linked, and must not be archived along each and every single execution. In addition, they commonly lack the overall awareness of a workflow. Therefore, there is a gap between these node-local and hardware-close tools and the higher level interaction a user wants to have with a HPC system.

## III. Overarching Concept of Integrating Data Management Tools into HPC Workflows

There are a number of challenges that a user typically faces when accessing an HPC system. For instance, a HPC system should reduce the complex and heterogeneous storage architecture to a unified namespace to offer users a quick and easy overview of their data. In addition, the data management system (DMS) should optimize the usage of a tiered storage system to provide maximal performance during compute and uses durable and low-cost storage for cold data. It should also adhere to the FAIR principles and perform transparent provenance auditing to ensure reproducibility. All data within a flat namespace should be searchable by domain-specific, semantic metadata, ideally even with respect to globally enforced policies for data access.

First, we identify and discuss the abstract components and their features when interacting with a storage and compute infrastructure, such as an HPC system. Then we describe the status quo as an archetype for the standard interaction paradigm and our envisioned user-friendly and data-centric flow.

### *A. Components*

The components necessary when handing data and compute are as follows:

- Resources - these are raw storage, compute and network infrastructures such as compute nodes and object/file systems and their interconnect. They come with their own specification, i.e., what resource they actually provide and their characteristics.
- Resource management (compute) - this layer manages the usage of the resources by assigning compute jobs to available compute resources satisfying the requirements for the respective (parallel) jobs.
- Resource management (storage) - this abstract concept defines where to store certain data and provides the respective space on a storage system.
- Job specification - defines the scope of a compute job together with it's requirements and specification such that it can be executed.
- Program - a code that can be executed on the compute infrastructure, e.g., binary program or script.
- Software landscape - the ecosystem and environment provided by the platform that allows to prepare programs on the system.
- Workflow specification - defines how to execute jobs in order to achieve the overall data-processing goal.
- Data management plan - defines for any data inputs and data products the policies, data handling and such to enable the FAIR principles while the data sovereignity of the user is preserved.
- User interface - allows the user to interact with the system, e.g., to manage and interact with some or all of the above components and to upload/download data.
- Client - the computer system of the user, where the user interface(s) are accessed.

The way, how one can fulfill the previously specified requirements and implement the components depends on the way a user wants to interact with it and the system providing these capabilities, and the data flow involved. For example, either, a user connects to the HPC system, and uses it as the central contact point, or the interface of the DMS is used to manage the data processing on the HPC system remotely.

### *B. Interaction Paradigms*

On the most extreme scale, one can argue that there are three different kinds of users. For instance, tech-affine people who want to natively work on the HPC in a traditional command-line approach, users that utilize state-of-the-art compute-centric tools, or those, who ideally only want to work with the interface of their domain-specific DMS.

*1) Traditional Paradigm:* In the traditional approach, the user interface is a shell (such as bash) on a login node of the cluster and the client is an SSH-enabled program that the user runs on their Desktop/Laptop. There exists no data management plan, the user thinks about how to manage data, therewith, manually performs the resource management for storage, identifies how to map output data to files (influenced by the applications) and directory structures, and utilizes the available parallel file systems. Also, the user manually prepares programs s/he wants to use by downloading the necessary codes on the machine and ensuring it works with the system architecture and software environment that is deployed on the HPC system. The wider software landscape on the HPC system was prepared by data center staff but libraries can be extended by the users in order to create meaningful programs. In most cases, workflows are not explicitly specified but manually invoked. The resource management of the compute resources is provided by tools such as Slurm. The job specifications are (bash) scripts that are invoked - they define the compute requirements. Such scripts are submitted to Slurm which decides how to map and schedule them on the available compute resources. These steps are basically manually set up, requiring a scientist to think about how the experiment should be conducted and then documented (if at all) in a lab notebook or scripts that do some of the work. Potentially, workflow tools such as Snakemake are utilized to specify dependencies between tasks and to automate dependencies between tasks. This is not only error-prone, but any change to the environment requires the user to modify the experimental setup and perform the steps again. We consider this the most typical interaction with the HPC system archaic.

*2) Compute-Centric Paradigm:* In a Compute-centric approach, a user would connect to the HPC frontend as usual. In the simplest form, a user would delegate the job of maintaining a data catalog and staging the selected input data to a DMS tool. This workflow is depicted in Figure 1.

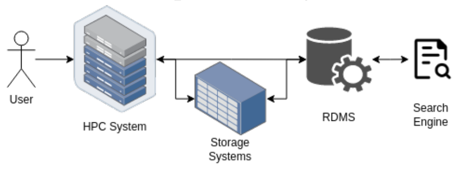

Fig. 1. HPC-centric flow

Here, in order to get access to the requested input data, a user would formulate a domain-specific, semantic search query and send this request to the DMS. Usually, a DMS would use a dedicated database or search engine to filter the requested data. The data is loaded into the running code of the user. This could either require a dedicated data transfer to a pre-configured storage target, or the HPC system and the DMS are already working on the same storage system. When looking at different established systems, compare, the user is typically responsible for lineage recording and enforcing reproducibility. Therefore, these solutions typically only assist users to manage and organize their data but do not free them from the burden of efficient IO and working in agreement with good scientific practice.

*3) Use-Case-Centric Paradigm:* The opposite way to integrate a DMS into an HPC workflow, is to use the DMS as the user frontend, see Figure 2.

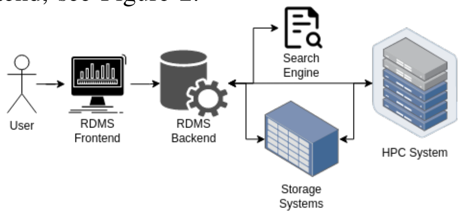

Fig. 2. DMS-centric flow

This DMS-provided user interface can be used to query and select input data, define a compute job, and submit this job to an HPC system, without the need to extra login to the HPC system or transfer data explicitly. This functionality requires a communication channel, between a remote DMS and an HPC system. Additionally, the DMS needs to be able to work with the individual resource manager, of each HPC system. The advantage of this approach lies in the capability to perform transparent lineage recording and can guarantee reproducibility. That is, because the DMS have complete control over the input data and the processes which run on them, using thorough logging methods is enough to ensure reproducibility. Similar to the HPC-centric use case, storage tiering is hard to support. In some existing implementations [17] data staging is explicitly required for each task execution.

## C. Data Flow

These two scenarios also differ in their data path, i.e. in the storage systems involved in the data management and data processing.

*1) Compute-Centric:* In the HPC-centric use case, a user accesses data through their respective, native interface, e.g. through the library functions of their respective programming language, or as an input parameter of their program which they want to run. That means, that only data is available which is directly accessible from the HPC system, and data transfers, e.g., for better performance, have to be done manually. In

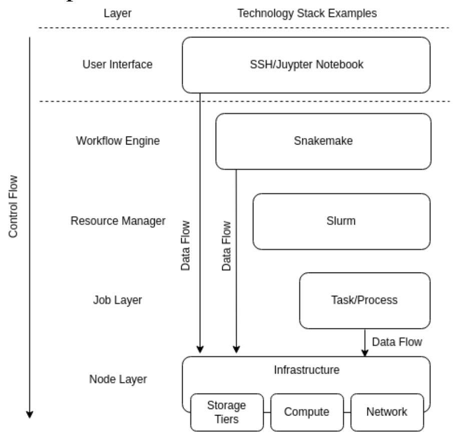

Fig. 3. Compute-centric flow

Figure 3 a layered diagram is shown which shows the possible data staging strategies. Within the user interface, e.g. ssh or a jupyter notebook, a user can explicitly copy/stage data on non-node-local storage. Within this layer, this has to be deliberately done. Assuming that a workflow engine, like snakemake, is used, data can be staged on non-node-local storage in a more automated and transparent way in the form of a dedicated workflow step. These two options can be considered asynchronous data staging since this will not lead to stalling times on the compute infrastructure. Synchronous data staging happens on the Job Layer, where a process first has to access data on a slow storage tier and stage it, and in this case even on a very fast node-local storage tier, before it can continue to process the data. Since the node-local storage is typically only available during the resource reservation, which is managed by the resource manager, e.g. Slurm, a user has to ensure that the overall process run on that node, does not only stage data, but also archives it once it is done. The entire data staging and IO optimization is therefore solely the user's responsibility.

*2) DMS-Centric:* Within the DMS-centric approach, a user is generally not interested to access the data directly, e.g. with a suitable library into self-written code. Instead, they are rather interested to have the complexity of running their job abstracted away. For this strategy, a layered diagram is

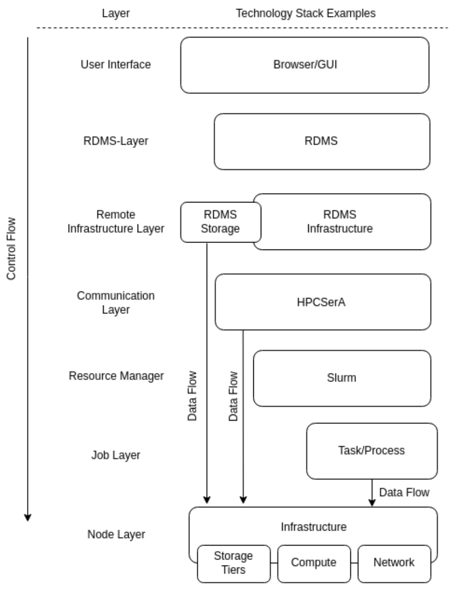

Fig. 4. DMS-centric flow

shown Figure 4, depicting the data flow. Here, a user access the DMS via a browser, or a DMS-specific graphical user interface (GUI). Within this interface, a user triggers the execution of a workflow, or a single analysis step on the selected input data. Often, these DMS are deployed in a cloud environment and have their own storage layer. Depending on whether this storage tier can be integrated into the HPC system, there are different strategies for data access. Either one can asynchronously or synchronously fetch data from the DMS within a dedicated data mover process and stage it either node-local, in the case of a synchronous data transfer, or non-node-local in the case of an asynchronous data transfer. For this purpose, the data mover process would either be granted access to the DMS storage with respect to the user's permissions, or the DMS can provide an endpoint, for instance, a REST endpoint, from which the process can fetch the required data. Since generally there is a communication layer, like HPCserA, required, to access the resources of an HPC system from the outside, this can also be used to asynchronously fetch data from the DMS. Lastly, a dedicated mover process can also fetch the data synchronously from the DMS on the compute node itself. This means, that the entire data movement and staging strategy is solely in the hands of the admins of the DMS, where the corresponding functionality is implemented and configured.

## D. Control Flow

Similar to the aforementioned data flow, there also exists a control flow, as can be seen by the left arrow in Figure 3 and Figure 4. The control flow is initially triggered by the user within the user interface and is from there passed down to the final task running on a node. From this upmost layer, the control path goes down via an optional workflow layer to the resource manager where the tasks get mapped on the actual hardware, in case of the HPC-centric view. In the DMS-centric view, the control flow gets even more abstracted, since the user input recorded by the user interface has to first pass through DMS layer, where the user request gets initially processed and mapped on the DMS infrastructure. Since this DMS system is completely disjunct from the HPC system, a dedicated communication layer, like HPCSerA, is required to bridge those two systems. On the HPC system, the individual tasks are again mapped to the nodes in the infrastructure layer via the resource manager.

## E. Analysis

To summarize the previous discussion about the different user interaction paradigms, Table I compares the characteristics of the individual components – ignore the column Governance-Centric for now.

The responsibility for one of the defined components and features are either the user, i.e., a manual process, semi-automatic - thus aiding the user (potentially following a specification), or fully automated. User-specific means it depends on the skill of the user. The resulting differences in the degree of automation can be illustrated best if we look at the interfaces a user can use to interact with the processes and data. Traditionally, only ssh connections are supported, whereas at least for the process interaction in the compute-centric paradigm, some interactions may take place via a web interface. In the use case-specific paradigm typically only a web interface is available that hides the HPC system and all internal processes.

The resource handling requires a lot of manual interaction of the users in the traditional and the compute-centric concept, while it is completely automated in the use case-centric approach based on configurations provided by the admins of the specific system.

A similar pattern can be seen in the characteristics of the task-related components. Here, the user experience with the HPC system evolves from a completely manual interaction to a partially automated or guided system in the compute-centric context, where already some low-level programming tools for workflow orchestration, data selection, and containers for dependency management are used. However, the program and data management rely still on manual work and are therefore potentially error-prone. On the other side, the use case-centric system fully automates these steps. Again, all task-related interactions are fully automated by the use case-specific system.

These intrinsic characteristics have different advantages and disadvantages. The traditional HPC usage paradigm relies heavily on manual work by the user to achieve a reasonable performance. Also the data management, the integration of storage and compute, and therefore the overall reproducibility is very much exposed to user errors. However,

TABLE I
COMPARISON OF DIFFERENT HPC USER INTERACTION PARADIGMS

| Characteristics | Traditional | Compute-Centric | **Governance-Centric** | Use Case-Centric |
|---|---|---|---|---|
| Resources (Compute) | Auto | Auto | Auto | Auto |
| Resources (Storage) | Manual | Manual | Auto | Auto |
| Res. Mgmt (Compute) | Semi-Auto | Semi-Auto | Auto | Auto |
| Res. Mgmt (Storage) | Manual | Manual | Auto | Auto |
| Job spec | Manual | Semi-Auto | Semi-Auto | Auto |
| Program | Manual | Manual | Semi-Auto | Auto |
| Software land | Provided | Provided/User-Container | Provided/User-Container | Provided |
| Workflow spec | Manual | Semi-Auto | Semi-Auto | Auto |
| DMP | Manual | Manual | Semi-Auto | Tool-specific |
| User interface | SSH | SSH+Web | Web+SSH | Web |
| User interface (Data) | SSH | SSH | Web+SSH | Web |
| Client | SSH | SSH+Browser | Browser+SSH | Browser |
| Performance | User-specific | User-specific | ++ | + |
| Data management | - | - | ++ | Tool-specific |
| Integration | – | 0 | ++ | ++ |
| Reproducibility | – | + | ++ | Tool-specific |
| Flexibility | ++ | ++ | + | – |

this enables the highest level of flexibility. The compute-centric paradigm improves this by utilizing the discussed semi-automated components and hereby improves the integration and reproducibility. The use case-specific systems will most likely have reasonable, but not custom-made, configurations to achieve good performance - here the interaction with data is challenging as the upload/download via Web-frontend limit performance. The current challenge is to unify the concepts of the compute-centric and the use case-centric paradigms and combine the advantages of these worlds.

## IV. GOVERNANCE CENTRIC ARCHITECTURE

Our goal is to expand upon the existing concepts to provide a novel, unified view of processes and data in order to improve the user experience on HPC systems. Here, the metrics for the user experience, i.e. performance, data management, integration, reproducibility, and flexibility, all basically boil down to the question of where the data is located and how they are linked. This has to be tackled simultaneously in two directions: first, an additional integration layer above the resource manager (compare Figure 3 and Figure 4) is required. This layer has to provide an integrated and unified namespace to the users. Secondly, an information flow, which is directed in the opposite direction as the control flow in Figure 3 and Figure 4, is required. Although this can be achieved with available auditing tools, see Section II-D, there is no concept for a tool that processes the incoming information and hereby makes it actionable. The advantage of an actionable information flow compared to the existing systems is that the information is utilized to create a desired, predefined state, and not just create yet another piece of data a user has to manage manually.

To this end, we propose the governance-centric interaction paradigm that aids the users and automizes the integration of data and compute. In Table I, we have identified the required degree of automation to bridge the gap between the compute-centric and the use-case-centric paradigms. Resource management should be fully automated to achieve the highest degree of integration of data and compute. The task-specific components should be semi-automated to guide the user in managing the data, working reproducibly, and ensuring that a predefined, ideal state is reached while allowing as much flexibility as possible. Similarly flexible should be the interface, to allow interaction from both user groups.

We believe professional and proper data management requires users to define an *experimental description* at the beginning of a project consisting of a workflow linking data sets and compute tasks and a data management plan for the respective input/output data. Then the user has to initially *modify their tasks*, e.g., job scripts, to allow linking of the tasks and their data products to the workflow and also to generate descriptive metadata for the data sets. Building upon the previous discussion, the goal is to not only use the workflow as an abstract concept that users may informally follow but rather enforce its usage. The *implications of the design* are that the HPC system can exploit the information to perform many previously manual tasks automatically and fulfill our goals.

For instance, to automatically receive and process information about input data and artifacts created during task execution, or enforcing archival/deletion policies defined in the DMP. To unify both user groups, the ingest of results into a DMS along with all required metadata including lineage information has to be one of the supported features.

To explain this idea in more detail, the experimental description shall be a user-defined and machine-readable workflow description that contains information about the data flow, the tasks which process these data sets and create artifacts, and further optional information like access policies or the IO intensity of each task. This means that, for every task a user wants to schedule via the resource manager, this task has to be linked to a specific workflow step within the experimental description at job submission time. Thus, each and every submission of a job on a HPC system becomes one concrete invocation of the abstract task description within the experimental workflow linked to data in the DMP.

### A. Experimental Description

Specifically, in data-driven projects, it is common that there is not a single task, but that the entire processing consists of multiple steps which are concatenated into a workflow. Therefore, a user has to provide a simple graph, compare Figure 5, connecting input and output data via tasks as a workflow description. This workflow could represent a weather prediction, where each cycle represents the simulation of the next hour (in the future); Dataset2 is the initial conditions while Dataset1 holds the model. Manual steps in the workflow are explicitly annotated as they require data to be accessible.

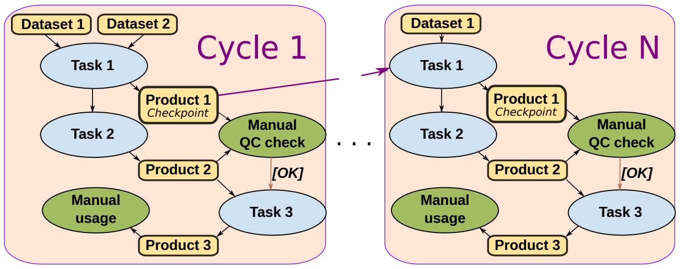

Fig. 5. High-level view of a workflow.

Within this workflow definition, general policies can be defined, e.g. where and when data should be archived, how long artifacts should be kept on hot storage if a manual inspection is required, what accompanying metadata are required, or if input data can be altered. The archiving of data should explicitly support remote DMS as a target, to integrate HPC systems with remote DMS, and similarly, data within a remote DMS should also serve as possible input data. The required data mover tools have therefore to be integrated as dependencies into the DMP. In addition, the users can provide information about the expected IO profile, aiding the proposed DMP tool to find the best storage tier based on heuristics configured by the HPC admins. Based on further metrics, like the available bandwidth of a remote DMS and the HPC system, or the amount of data, the DMP tool can also determine, whether data should be staged synchronously, i.e. during compute time, or asynchronously, i.e. as a dedicated, dependent step before the compute task starts.

*Describing Datasets:* The user can and should add further information to the data sets which are getting processed or created. To improve the findability a user should provide domain-specific metadata, or define a task to extract those. The required, and optional domain-specific metadata fields can be defined in the DMP. For instance, in our figure, Product 2 may be characterized by the date/time of the weather prediction and each product could be tagged with the model configuration settings. This can even ensure a homogeneous metadata quality across a larger group working on a joint project. In addition, the data life cycle should be defined, i.e. what is the retention time, what are the deletion policies. To meet the data governance policies required by the user, additional aspects such as access control must be defined to prevent unwanted data leakage.

## B. Modifying Tasks

Compute jobs on HPC systems are dispatched to the actual resources using resource managers such as Slurm. On this level, a job has to be prepared, annotated, and linked with the workflow. The user annotations should specify the task within the defined workflow, which is to be executed. In addition, a user can further restrict and specify the input data. Here, the largest change compared to the traditional HPC interaction paradigm becomes apparent: Instead of working with explicit files, a user rather works with datasets defined by metadata. For instance, a user specifies the input data either based on domain-specific metadata or simply due to the link in the DMP. Therefore, the actual storage location is abstracted from the user. The actual data directory, which a program still needs to specify in API calls, can be automatically exported via environment variables or generated via support tools in the job script. Before reserving dedicated compute resources, the proposed DMP tool decides to use either synchronous or asynchronous data staging and stages the data respectively.

One key requirement in science is reproducibility. In a first step, this requires at least sufficient provenance information to allow for retrospective comprehensibility of the lineage of the resulting artifacts. One important element to retrospectively comprehend an HPC job is the run script used for batch processing. This can be automatically archived along with the artifacts by the proposed DMP tool. However, these batch scripts, which contain the actual compute job to be run in the form of a shell script, can have multiple ambiguities. One simple example of this would be the execution of an interpreted script, e.g. a *Python* script. Here, one would have a simple line within the batch-script which would look similar to:`$ python my_script.py`
The challenge within this call is to track differences between multiple invocations of this script, where the content of *my_script.py* has been changed. For this, three different high-level modi are proposed.

The recommended way is to use a Git repository, where changes in code are properly tracked. In this case the DMP tool checks in the directory of the script and saves the information about the Git repository and the used Git commit hash so that this information can be stored in the metadata of the created data products. The DMP tool will create a dedicated sidecar file for this metadata in the output directory. The usage of version control systems can, and should, also be part of the required specifications within the DMP.

Alternatively, if no Git repository is set up, the batch script is parsed and untracked dependencies in the user namespace, like a Python script, are tried to be identified and archived along with the artifacts. Since this method is potentially more error-prone compared to a proper version control system, like Git, it is not recommended, but should still offer a better chance for retrospective comprehensibility when compared to other strategies. These dependencies will be listed in the before-mentioned sidecar file, and are archived alongside it. One important distinction to make is the use of containers. In this case, the container image should be archived and linked to the sidecar file. Of course, utilizing provenance-specific tools, as discussed in Section II and translating them to the required standard in the sidecar file, is also an option to explore.

The third option is that users compose the sidecar file by adding code to the batch script, where this provenance information are provided. This can be added to the directory where the output, which should be archived, resides. This file will also override information that was automatically tried to extract in the previous step.

### C. Implications of the Design

This presented design has different positive implications on the user experience that we summarize in Table I.

*a) Integration:* First of all, the abstraction of files on storage towards more high-level data sets achieves the tight integration of storage and compute. Abstracting the storage from the data will motivate users to use proper metadata management systems and establish a data catalog, instead of encoding information into file paths.

*b) Performance:* Since users are only working with datasets and not with a filepath anymore, HPC admins can configure data placement strategies, therefore relieving this burden from the users and optimizing the performance.

*c) Reproducibility:* Since compute tasks, their input, and resulting data are strongly linked with each other, the lineage of artifacts is much more comprehensible and less subjected to user errors. Utilizing further tools like containers and a version control system will ensure full reproducibility, which is integrated into this paradigm by design because this is just another policy in the defined data governance, which will be enforced for the users.

*d) Enforcing DMP:* Although the general idea of using data management plans in HPC is far from new, the novel advantage of this particular tool is that it can be enforced. There are various ways to achieve this goal. A naive approach compatible with existing systems is to use a cronjob that reads in the workflow and task definition files, which a user has provided, and compares the specified, desired state of the storage systems of the user against the actual state at hand. If new output data are detected and the required sidecar file for the necessary metadata is available, the output data is handled as specified. However, if the required sidecar file is not, or only with insufficient content provided the user will be reminded to provide the missing information after a specified grace time. Similarly, if data is detected which can not be matched to the dataset specification in the workflow definition, an error or warning can be raised to the user as such unclassified data shall not exist. Thus, the DMP becomes actionable and hereby ensures a homogeneous system state in sync with the experimental description and user expectations.

## V. Conclusion and Future Work

In conclusion, we introduced the governance-centric interaction paradigm, which by design integrates storage and compute for the users. It relies on a researcher to define an experimental description and a DMP for the datasets at the beginning, something that good scientific practice requires anyhow. This will allow the system to perform various tasks on behalf of the user and increase overall automatization. Ultimately, the burden of performance optimization can be shifted partially from each user to data center operators. Furthermore, the abstraction of files to data sets allows the seamless integration of a DMS. Since this paradigm seamlessly links data to compute tasks, it ensures retrospective comprehensibility and reproducibility by design.

We are in the process of developing tools and an environment where this vision is implemented. In future work, this concept will be evaluated on specific use cases. In addition, synthetic benchmarks will be used to evaluate the proposed concept of storage and compute integration with other tools offering a unified namespace across a tiered storage system.

## Acknowledgement

We gratefully acknowledge funding by "Nationales Hochleistungsrechnen" and BMBF under 01—S22093A.

## References

[1] M. D. Wilkinson *et al.*, "The fair guiding principles for scientific data management and stewardship," *Scientific data*, vol. 3, no. 1, pp. 1–9, 2016.

[2] P. Corbett *et al.*, "Overview of the mpi-io parallel i/o interface," *Input/Output in Parallel and Distributed Computer Systems*, pp. 127–146, 1996.

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

[4] F. B. Schmuck and R. L. Haskin, "Gpfs: A shared-disk file system for large computing clusters." in *FAST*, vol. 2, no. 19, 2002, pp. 231–244.

[5] V. Korol *et al.*, "Introducing viking: A novel online platform for multiscale modeling," *ACS omega*, vol. 5, no. 2, pp. 1254–1260, 2019.

[6] M. T. Nelson *et al.*, "Namd: a parallel, object-oriented molecular dynamics program," *The International Journal of Supercomputer Applications and High Performance Computing*, vol. 10, no. 4, pp. 251–268, 1996.

[7] D. Van Der Spoel *et al.*, "Gromacs: fast, flexible, and free," *Journal of computational chemistry*, vol. 26, no. 16, pp. 1701–1718, 2005.

[8] D. S. Marcus, T. R. Olsen, M. Ramaratnam, and R. L. Buckner, "The extensible neuroimaging archive toolkit: an informatics platform for managing, exploring, and sharing neuroimaging data," *Neuroinformatics*, vol. 5, pp. 11–33, 2007.

[9] R. L. Harrigan *et al.*, "Vanderbilt university institute of imaging science center for computational imaging xnat: A multimodal data archive and processing environment," *NeuroImage*, vol. 124, pp. 1097–1101, 2016.

[10] K. Shvachko, H. Kuang, S. Radia, and R. Chansler, "The hadoop distributed file system," in *2010 IEEE 26th symposium on mass storage systems and technologies (MSST)*. Ieee, 2010, pp. 1–10.

[11] V. K. Vavilapalli *et al.*, "Apache hadoop yarn: Yet another resource negotiator," in *Proceedings of the 4th annual Symposium on Cloud Computing*, 2013, pp. 1–16.

[12] M. Milligan, "Interactive hpc gateways with jupyter and jupyterhub," in *Proceedings of the Practice and Experience in Advanced Research Computing 2017 on Sustainability, Success and Impact*, 2017, pp. 1–4.

[13] K.-K. Muniswamy-Reddy, D. A. Holland, U. Braun, and M. I. Seltzer, "Provenance-aware storage systems." in *Usenix annual technical conference, general track*, 2006, pp. 43–56.

[14] D. Dai, Y. Chen, P. Carns, J. Jenkins, and R. Ross, "Lightweight provenance service for high-performance computing," in *2017 26th International Conference on Parallel Architectures and Compilation Techniques (PACT)*. IEEE, 2017, pp. 117–129.

[15] F. Chirigati, R. Rampin, D. Shasha, and J. Freire, "Reprozip: Computational reproducibility with ease," in *Proceedings of the 2016 international conference on management of data*, 2016, pp. 2085–2088.

[16] J. Lofstead, J. Baker, and A. Younge, "Data pallets: containerizing storage for reproducibility and traceability," in *High Performance Computing: ISC High Performance 2019 International Workshops, Frankfurt, Germany, June 16-20, 2019, Revised Selected Papers 34*. Springer, 2019, pp. 36–45.

[17] S. Bingert, C. Köhler, H. Nolte, and W. Alamgir, "An api to include hpc resources in workflow systems," in *INFOCOMP 2021: The Eleventh International Conference on Advanced Communications and Computation*, 2021, pp. 15–20.