Data reduction in scientific workflows using provenance monitoring and user steering

Renan Souza, Vitor Silva, Alvaro Luiz Gayoso de Azeredo Coutinho, Patrick Valduriez, Marta Mattoso

Data Reduction in Scientific Workflows using Provenance Monitoring and User Steering

Renan Souza[a,b], Vítor Silva[a], Alvaro L G A Coutinho[a],
Patrick Valduriez[c], Marta Mattoso[a]

[a]*COPPE/Federal University of Rio de Janeiro, Brazil*
[b]*IBM Research, Brazil*
[c]*Inria and LIRMM, Montpellier, France*

Abstract

Scientific workflows need to be iteratively, and often interactively, executed for large input datasets. Reducing data from input datasets is a powerful way to reduce overall execution time in such workflows. When this is accomplished *online* (*i.e.*, without requiring the user to stop execution to reduce the data, and then resume), it can save much time. However, determining which subsets of the input data should be removed becomes a major problem. A related problem is to guarantee that the workflow system will maintain execution and data consistent with the reduction. Keeping track of how users interact with the workflow is essential for data provenance purposes. In this paper, we adopt the "human-in-the-loop" approach, which enables users to steer the running workflow and reduce subsets from datasets online. We propose an adaptive workflow monitoring approach that combines provenance data monitoring and computational steering to support users in analyzing the evolution of key parameters and determining the subset of data to remove. We extend a provenance data model to keep track of users' interactions when they reduce data at runtime. In our experimental validation, we develop a test case from the oil and gas domain, using a 936-cores cluster. The results on this test case show that the approach yields reductions of 32% of execution time and 14% of the data processed.

Keywords
Scientific Workflows; Human in the Loop; Online Data Reduction; Provenance Data; Dynamic Workflows.

1. Introduction

Scientific Workflow Management Systems (SWMSs) with parallel capabilities have been designed for executing data-intensive scientific workflows, or scientific workflows for short, in High Performance Computing (HPC) environments. A typical execution may involve thousands of parallel tasks with large input datasets [1]. When it is iterative, the workflow is repeatedly executed for each element of an input dataset. The more the data to process, the longer the workflow may take, which may be days depending on the problem and HPC environment [2]. Configuring a scientific workflow with parameters and data is hard. Users[1] typically need to try several input data or parameter combinations in different workflow executions. These trials make the scientific experiment take even longer. The obvious solution to improve performance in HPC is through parallel processing. However, reducing the input data to be processed can be also very effective to reduce workflow execution time [3].

In scientific workflows, the total amount of data is large, but not necessarily the entire input dataset has relevant data for achieving the goal of the workflow execution. This is

[1] Our target-user profile is a computational scientist, who is expert in domain-specific systems (*e.g.*, bioinformatician, computational physicist, or engineer).

particularly true when a large parameter space needs to be processed in parameter sweep workflows. There may be slices of the parameter space that have no (or little) influence on the results and thus, as with a "branch and bound" optimization strategy, can be bounded before its evaluation. A similar scenario occurs when the workflow involves a large input dataset. When users can actively participate in the computational process, practice frequently referred to as "human-in-the-loop", they may analyze partial result data and tell which part of the data is relevant for the final result [4]. Then, based on their domain knowledge, users can identify which subset of the data is not interesting and thus should be discarded from the execution by the SWMS, thereby reducing execution time.

Data reduction can be accomplished in at least three different forms. First, it can be done before the execution starts. However, in most complex scenarios, the high number of possibilities make it impossible to know beforehand the uninteresting subsets, without any prior execution. Furthermore, not only the initial dataset can be reduced, but also the intermediate data generated by the workflow, since the activities composing it continuously produce significant amounts of data that are consumed by other activities. A second form of data reduction is online. When the SWMS allows for partial result data analysis, the user may analyze the partial data, find which slice of the dataset is not interesting, and reduce the dataset online. We use the term *online* when user is able to inspect the workflow execution, analyze partial and execution data, and dynamically adapt workflow settings while the workflow is running (*i.e.*, at runtime). The third form of data reduction is by stopping the execution, reducing the data offline, and then resume with the reduced dataset. Because of the difficulty in defining the exploratory input dataset and the long execution time of such workflows, users frequently adopt the third form. However, in the offline form, the SWMS is not aware of the changes, and the results with one workflow configuration are not related to the others. Therefore, this is generally more time-consuming, there is no control or registration of user interactions, and the execution may become inconsistent [2].

Online data reduction has obvious advantages but introduces several challenges related to computational steering in HPC environments [4]. First, because of the complexity of the scientific scenario and the huge amount of data, users do not know beforehand which data subset should be kept or removed. Identifying these subsets involves relating input data to intermediate data and final results. Also, if users cannot actively follow the result data evolution online, in particular, domain data associated to execution and provenance data (history of data derivation), they can be driven to misleading conclusions when trying to identify the uninteresting data subset. Second, if they can find which subset to remove and try to remove it, the SWMS must allow for such online reduction and guarantee that the operation will be done consistently. Otherwise, it can introduce anomalous data, with no control over data elimination, data redundancy, or even execution failure. Third, in a long run, there may be more than one interaction, each removing more subsets, at different times. If the SWMS does not keep track of the user's actions, it negatively impacts the results' reproducibility and reliability. Although data reduction is not new in SWMSs [3], the problem of doing this online, steered by users, while maintaining data provenance has not been addressed before.

To address these challenges, we propose a data reduction approach. The key idea is to consider input datasets as sets of input data elements that can be manipulated and related to their following data elements along the dataflow generation. Provenance data management is at the core of the approach. In addition to the traditional advantages of managing provenance data in scientific workflows (*i.e.*, reproducibility, reliability, and quality of result data) [5], online provenance data management eases interactive domain data analysis [6,7]. Such analysis helps finding the data subset to be removed. Moreover, the SWMS must guarantee data consistency in the execution before and after the reduction, and keep track of user steering actions to maintain provenance of adaptations.

Chiron is a SWMS that implements a data-centric approach. It has successfully been used to manage scientific workflow applications in domains such as bioinformatics [6], computational fluid dynamics [2], and astronomy [7]. However, Chiron does not control changes in input datasets, including removing a subset. To implement our data reduction approach in Chiron, we add new operators and modules to enable users to reduce sets of input data for workflow activities online, and maintain consistency of the provenance of the removed data. We take advantage of a distributed in-memory database system (MySQL Cluster) in a version called d-Chiron that is significantly more scalable than Chiron [8], to address consistency issues with respect to data reduction. We make the following contributions:

- A mechanism coupled to d-Chiron SWMS for online input data reduction, which allows users to remove data subsets at runtime. It guarantees that both execution and data remain consistent after reduction.
- An extension to a provenance data diagram (which is W3C PROV compliant) to maintain the history of user adaptations when users reduce data online.
- A module to track provenance of human adaptation when users reduce data online. The mechanism collects provenance of human-adaptation data when data reductions are done in the datasets being consumed by the workflow. The provenance data is inserted online in the wf-Database, which implements the extended data diagram.

This paper is a major extension of [9], which introduces the initial ideas of online input data reduction in scientific workflows. In this paper, we extend [9] with: (i) a formalization of the core concepts of the solution; (ii) a formal definition of the user steering operator to reduce input data; (iii) practical examples of how other real-world scientific workflows (other than the one used in the experimental validation), such as SciPhy [10] and Montage [11], could benefit from our solution; (iv) explanation about how consistency is tackled in the approach, and a detailed description about how we implement it; (v) details about how users use the system; (vi) the exploration of different aspects of the benefits of our solution through a broader set of experiments; and (vii) more related work.

Paper organization. Section 2 gives our motivating example. Section 3 describes the background for this work. We present our main contribution in Section 4 and its implementation in Section 5. Section 6 shows how users use the system, Section 7 presents an experimental validation, Section 8 shows related work, and Section 9 concludes.

2. Motivating Case-study in the Oil and Gas Industry

In ultra-deep water oil production systems, a major application is to perform risers' analyses. Risers are fluid conduits between subsea equipment and the offshore oil floating production unit. They are susceptible to a wide variation of environmental conditions (*e.g.*, sea currents, wind speed, ocean waves, temperature), which may damage their structure. The fatigue analysis workflow adopts a cumulative damage approach as part of the riser's risk assessment procedure considering a wide combination of possible conditions. The result is the estimate of riser's *fatigue life*, which is the length of time that the riser will safely operate. The Design Fatigue Factor (DFF) may range from 3 to 10, meaning that the riser's fatigue life must be at least DFF times higher than the service life [12].

Sensors located at the offshore platform collect external conditions and floating unit data, which are stored in multiple raw files. Offshore engineers use specialized programs (mostly simulation solvers) to consume the files, evaluate the impact of environmental loads on the risers in the near future (*e.g.*, risk of fractures), and estimate the risers' fatigue life. Figure 1 shows a scientific workflow composed of seven piped programs (represented by workflow activities) with a dataset in between, forming a flow of sets of data elements within linked tasks. The **<<Stereotypes>>** and dataflow concepts are explained in Section 3.1.

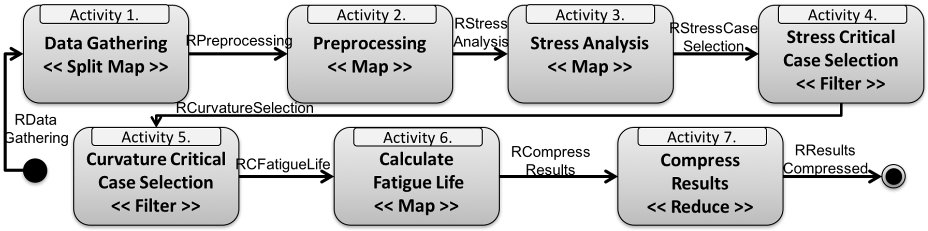


Figure 1. Risers Fatigue Analysis Workflow.

Each task of `Data Gathering` (Activity 1) decompresses one large file into many files containing important input data, reads the decompressed files, and gathers specific values (environmental conditions, floating unit movements, and other data), which are used by the following activities. `Preprocessing` (Activity 2) performs pre-calculations and data cleansing over some other finite element mesh files that will be processed in the following activities. `Stress Analysis` (Activity 3) runs a computational structural mechanics program to calculate the stress applied to the riser. Each task consumes pre-processed meshes and other important input data values (gathered from first activity) and generates result data files, such as histograms of stresses applied throughout the riser (this is an output file), and stress intensity factors in the riser and principal stress tensor components. It also calculates the current curvature of the riser. Then, `Stress Critical Case Selection` (Activity 4) and `Curvature Critical Case Selection` (Activity 5) calculate the fatigue life of the riser based on the stresses and curvature, respectively. These two activities filter out results corresponding to risers that certainly are in a good state (no critical stress or curvature values were identified). Those cases are of no interest to the analysis. `Calculate Fatigue Life` (Activity 6) uses previously calculated values to execute a standard methodology [12] and calculate the final fatigue life value of a riser. `Compress Results` (Activity 7) compresses output files by riser.

Most of these activities generate result data (both raw data files and some other domain-specific data values), which are consumed by the subsequent activities. These intermediate data need to be analyzed during workflow execution. More importantly, depending on a specific range of data values for an output result data (*e.g.*, fatigue life value), there may be a specific combination of input data (*e.g.*, environmental conditions) that are more or less important during an interval of time within the workflow execution. The specific range is frequently hard to determine and requires a domain expert to analyze partial data during execution. For example, an input data element for Activity 2 is a file that contains a large matrix of data values, composed of thousands of rows and dozens of columns. Each column contains data for an environmental condition and each row has data collected for a given time instant. Each row can be processed in parallel and the domain application needs to consume and produce other data files (on average, about 14 MB consumed and 6 MB produced per processed input data element). After many analyses online, the user finds that, for waves greater than 38 m with frequency less than 1Hz, a riser fatigue will never happen. Thus, within the entire matrix, any input data element that contains this specific uninteresting range does not need to be processed. Therefore, by reducing the input dataset, the overall data processed and generated are reduced and thus the overall execution time. In this paper, we use this workflow as basis for our examples.

3. Background

It is known that scientific workflows are data-centric. Data management in scientific workflows is critical due to the inherent complexity of the scientific domain data and the HPC requirements, such as efficient exploitation of data parallelism. In this section, we provide the background for this work, which relies on a data-centric algebraic approach for scientific workflows [13,14]. It provides constructs, mechanisms, and conceptualizations which in essence aim at valorizing fine-grained elements of data flowing throughout the workflow activities, rather than just the chaining of tasks (*i.e.,* chaining of programs or processes). This enables building solutions for dynamic data analyses and even adaptations in the dataflow. This data-centric approach is used by several modern parallel systems, such as Swift [15], Apache Spark [16], and Panda [17], in order to exploit data parallelism. In Section 3.1, we describe the data-centric approach and in Section 3.2, we explain user-steered workflows, which together set the foundation for this work.

3.1 Data-Centric Algebraic Approach for Scientific Workflows

A scientific workflow is composed of a set of *activities*. An activity can be a program, a script or a function that consumes in parallel *input datasets,* computes, and produces *output datasets*. The output dataset of a certain activity can become an input dataset of another, defining a *data dependency* between these two activities. A dataset (either input or output) is a set of *data elements*. A data element can contain simple primitive data values (*e.g.*, integers, strings) or complex data objects (*e.g.*, matrices, finite element meshes, images). An input data element has the necessary data to be consumed by a workflow activity computation, or *task*. In a workflow execution, in addition to the data dependency between chained workflow activities, there may be thousands of independent tasks running in parallel within each activity, as in the Many Task-Computing paradigm [1]. Figure 2 illustrates a generic representation of a scientific workflow, where the dataset R_2 of output data elements of activity Act_1 is also a set of input data elements for activity Act_2.

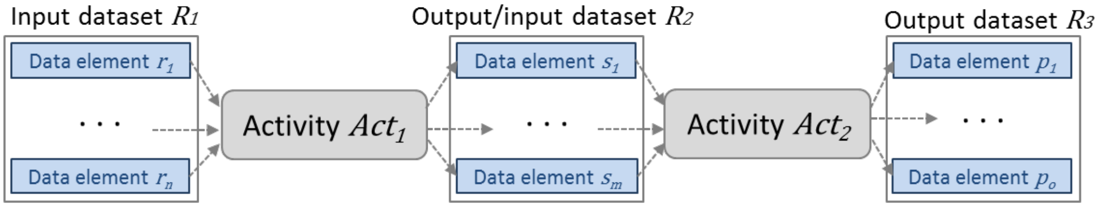


Figure 2. Data dependency between activities.

3.1.1 Data-centric Algebraic Notation

A workflow W is a chaining of activities $A = \{Act_1, \dots, Act_k\}, k = |A|$. An activity $Act_i \in A$ consumes input datasets R_i and produces output datasets R_{i+1}, which can be consumed, as input datasets, by an activity $Act_{i+1} \in A$, for all activities in A, forming a dataflow. This is presented in [7], inspired by concepts proposed by Ikeda *et al.* [17].

Not necessarily all workflow activities consume the data elements in the same way. For example, an activity Act_i may produce one output data element in R_{i+1} for each input data element consumed from R_i (*i.e.*, $1{:}\,1$ ratio between the cardinalities of the input and output datasets); and a chained activity Act_{i+1} may consume all n elements from R_{i+1} to produce a dataset R_{i+2} with a single data element ($n{:}\,1$ ratio). Workflow engines can highly benefit from this information to anticipate runtime optimizations of data parallel operations [14].

We distinguish between input and output datasets. *Input datasets* are composed of *input data elements* and are *consumed* by activities. *Output datasets* are composed of *output data elements* and are *produced* by activities. Then, let $R = \{R_1, \dots, R_r\}$, with $r = |R|$, be the set that contains all datasets (either input our output) for the activities in a workflow W. The data dependencies of the datasets form the dataflow. Considering these concepts, Ogasawara *et al.* [14] formalize the general form of the data-centric workflow algebra as:

$$R_{i+1} \leftarrow DT(Act_i, \{additional\ operands\},\ R_i),$$

where $DT \in \{Map, Reduce, Filter, MRQuery, SplitMap\}$ is a data transformation and $\{additional\ operands\}$ are additional operands that may be needed by some DT, such as *Reduce*. DT is a dataflow operator that determines how input data elements are transformed into output data elements, in particular, the ratio between number n of input data elements consumed and number m of produced output data elements in a workflow activity Act_i. For example, *Map* has 1:1 ratio, *Reduce* has $n{:}\,1$, *SplitMap* has $1{:}\,m$, filter has $1{:}\,(1|0)$, and *MRQuery* (*e.g.*, a join of two datasets) has $n{:}\,m$ ratio. Figure 3 shows a data-centric algebraic representation of the Risers Fatigue Analysis workflow shown in Figure 1.

$$\boxed{\begin{array}{l}
RPreprocessing \leftarrow SplitMap(DataGather, RPreprocessing)\\
RStressAnalysis \leftarrow Map(Preprocessing, RPreprocessing)\\
RStressCaseSel \leftarrow Filter(StressCaseSel, RStressAnalysis)\\
RCFatigueLife \leftarrow Filter(CurvatureCaseSel, RStressCaseSel)\\
RCompressResults \leftarrow Map(CalcFatigueLife, RCFatigueLife)\\
RResults \leftarrow Reduce(CalcFatigueLife, case, RCompressResults)
\end{array}}$$

Figure 3. Risers Fatigue Analysis workflow using the algebraic dataflow representation.

All datasets $R_i \in R$ have a predetermined data schema $\boldsymbol{\mathcal{S}}(R_i)$ = (*attribute*$_{i1}$: *datatype*$_{i1}$,…, *attribute*$_{iu}$: *datatype*$_{iu}$), with u = number of attributes of $\boldsymbol{\mathcal{S}}(R_i)$. The data elements of R_i follow the schema $\boldsymbol{\mathcal{S}}(R_i)$. For example, suppose that $R_1 \in R$ contains input environmental conditions and $R_2 \in R$ contains output fatigue life values, then possible schemas for these datasets could be: $\boldsymbol{\mathcal{S}}(R_1)$ = (*wind_speed: float, wave_frequency: float, air_temperature: float, humidity: float, mesh_path: string*) and $\boldsymbol{\mathcal{S}}(R_2)$ = (*fatigue_life: integer, histogram_path: string*). Thus, the structure of the data flowing between activities is enhanced. The user is also familiar with those domain terms. This not only contributes to the workflow engine for data parallelism, but also enables submitting structured queries on dataflow generation, which includes domain-data values in the flow.

3.1.2 Scientific Domain Data Management using the Algebraic Approach

In addition to efficiently managing data parallelism, the SWMS needs to manage the scientific domain data. Structured queries ease data analysis and the algebraic approach is useful here [2,4,7]. Two types of data are stored in those datasets: domain-specific values and other larger or more complex data. The algebraic approach enables expressing the scientific datasets in a structured way following a data schema $\boldsymbol{S}(R_i)$ with known data types. Often, scientific programs require the processing of other complex large datasets, such as large matrices, binary data, unstructured information, or other domain-specific data representations. We use paths that point to these data files on disk in the attributes of the data elements composing the datasets $R_i \in R$ whenever a file is read or written by an activity [7]. Some other domain-specific data values within those files can be extracted or generated based on the content of the file and represented as attributes in the data elements, also increasing the expressivity of the file descriptions [13,14]. These domain-specific quantities can be decisive for the domain so they need to be tracked. Since tracking all fine-grained data elements the user wants may be impractical due to the huge amount of data, the domain expert determines which domain-specific values need to be extracted and tracked. All these data management features enable users to query, analyze, and inspect the scientific raw data and the dataflow as the workflow is being processed.

3.2 User-steered Workflows

There are at least six aspects of computational steering in scientific workflows: interactive analysis, monitoring, human adaptation, notification, interface for interaction, and computing model [4]. Despite the importance of them all, the first three are essential and are the ones this work focuses on. Human adaptation is definitely at the core of computational steering. However, users will only know how to fine-tune parameters or which subset needs further focus if they can explore partial result data during a long-term execution. Thus, interactive analysis and monitoring are important to put the human in the loop. Online provenance data management in SWMSs is an essential asset to support all six aspects of computational steering in scientific workflows. In this section, we explain the three computational steering aspects explored in this paper and how the data-centric algebraic approach supports them.

3.2.1 Interactive Analysis

To allow for interactive analysis, the data-centric approach designs how fine-grained domain-specific data, workflow execution data, performance data, and provenance data are collected by the workflow engine. It also specifies how they are stored in a database (the wf-Database) to be queried by users. We address two aspects of workflow data that can be interactively analyzed: (A) domain dataflow and (B) workflow execution [4].

(A) Domain dataflow. To allow for domain dataflow interactive analysis, dataflow provenance data are stored in the wf-Database. The input and output data elements are continuously collected and stored in the wf-Database at each task execution at runtime. The output elements are linked to the inputs, so that the flow of data elements can be easily retrieved through provenance queries. This approach enables online fine-grained domain dataflow analysis [6] as well as the analysis of related domain data files through file flow relationships [7], as in Figure 4.

To exemplify some possible interactive queries, Table 1 has some typical analyses that are executed for the riser fatigue analysis workflow involving domain and provenance dataflow analysis. In an earlier work, we also observed similar query patterns for scientific data analysis in different domains [18]. For Queries $Q1$-$Q4$, the SWMS needs to store the history of the data elements generated in Activities 4 and 5 since the beginning of the flow, linking each element-flow in between. For example, environmental conditions ($Q1$) and

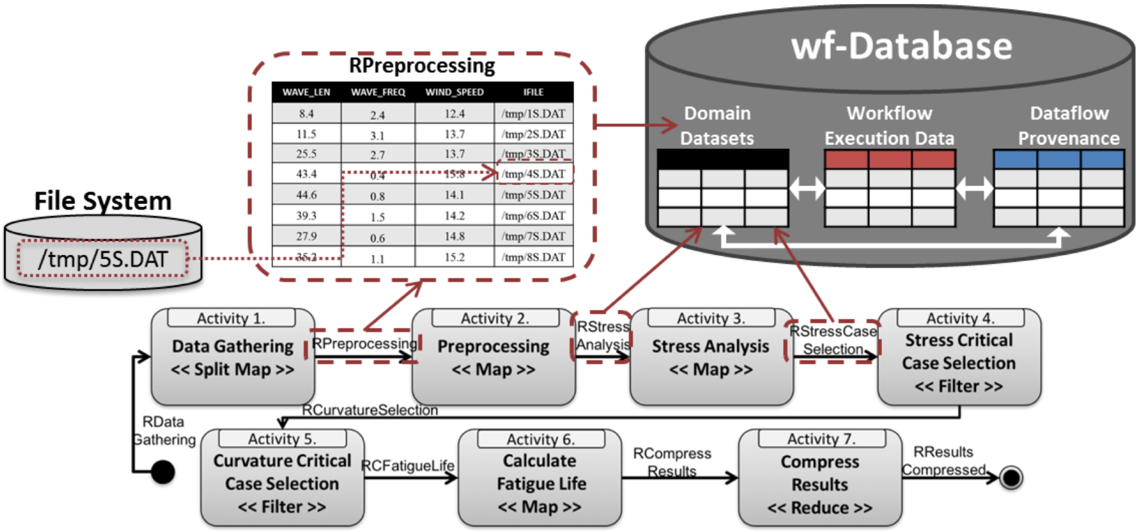


Figure 4. Scientific domain data management showing how the data elements flowing between activities are stored as datasets linked with workflow execution data and dataflow provenance in the wf-Database.

hull conditions ($Q2$) are obtained in Activity 1, and stress- and curvature-related values are obtained in Activities 4 and 5, respectively. To correlate output elements from Activity 4 or 5 to output elements from Activity 1, provenance data relationships are required.

Users can analyze the dataflow by running queries in the database query interface at any time during execution or using any application that connects to the database to plot data visualization. Without such structured query support, users need to look for files in their directories, open and analyze them, and try to do this analysis in an *ad-hoc* way. Frequently, they write scripts to search in these result files. They often interrupt the execution to fine tune input data and save execution time. This user behavior is observed not only in the oil and gas domain, but also in several other domains, such as bioinformatics, computational physics, and astronomy. More examples exploring how this data-centric approach enables querying domain dataflow together with provenance data to enhance online data analysis in scientific workflows can be found in [6,7,18].

Table 1. Domain dataflow provenance interactive queries.

Q1	What is the average of the 10 environmental conditions that are leading to the largest fatigue life value?
Q2	What are the water craft's hull conditions that are leading to risers' curvature lower than 800?
Q3	What are the top 5 raw data files that contain original data that are leading to lowest fatigue life value?
Q4	What are the histograms and finite element mesh files related when computed fatigue life based on stress analysis is lower than 60?

Table 2. Provenance and domain data linked to execution data.

Q5	Determine the average of each environmental conditions (output of `Data Gathering` – Activity 1) associated to the tasks that are taking more than the double of the average execution time of `Curvature Critical Case Selection` (Activity 5), grouping the results by the machines (hostnames) where the tasks of Activity 5 were executed.
Q6	Determine the finite element meshes files (output of Preprocessing – Activity 2) associated to the tasks that are finishing with error status.
Q7	List information about the 5 computing nodes with the greatest number of `Preprocessing` activity tasks that are consuming data elements that contain wind speed values greater than 70 Km/h.

B. Workflow execution. Lower level execution engine information, such as physical location (*i.e.*, virtual machine or cluster node) where a task is being executed, can highly benefit data analysis and debugging in large-scale HPC executions. Users may want to interactively investigate how many parallel tasks each node is running. Moreover, this approach eases debugging. Tasks run domain applications that can result in errors. If there are thousands of tasks in a large execution, how to determine which tasks resulted in domain application errors and what the errors were? Furthermore, performance data analysis is very useful. Users are frequently interested in knowing how long tasks are taking, how much computing resources (memory, CPU, disk IO, network throughput, etc.) are being consumed [19]. All this workflow execution data are important to be analyzed and can deliver interesting insights when linked to domain dataflow data. When execution data is stored separately from domain and provenance data, these steering queries are not possible or require combining different tools and writing specific analysis programs [7]. To support all this, the data-centric approach allows for recording parallel workflow execution data in a way that they can be linked to domain and provenance data. Table 1 shows some provenance queries for the Risers Analysis workflow that link workflow execution data to domain dataflow. Figure 4 shows how the domain-data elements flowing within the activities of Risers Fatigue Analysis workflow are managed as datasets in the wf-Database, linked to workflow execution data and dataflow provenance. Also, the datasets contain paths to raw data files on disk [7].

3.2.2 Human Adaptation

After users have analyzed partial data and gained insights, they may decide to adapt the workflow execution. It brings powerful abilities to users, putting the human in control of a scientific workflow execution. Many aspects can be adapted by humans, but very few systems support human-in-the-loop actions [4]. The human-adaptable aspects range from computing resources involved in the execution (*e.g.*, adding or removing nodes), to check-pointing and rolling-back (debugging), loop break conditions, reducing datasets, modification of filter conditions, and parameter fine-tuning.

Populating the wf-Database during workflow execution helps all these aspects. For example, in [20], it is shown that it is possible to change filter conditions during execution. Also, in [2], a data-centric algebraic approach is proposed to adapt loop conditions of iterative workflows (*e.g.*, modify number of iterations or loop stop conditions). These works show that adaptations can significantly reduce overall execution time, since users are able to identify a satisfactory result before the programmed number of iterations. Prior to this work, no work has been developed to tackle user-steered data reduction online taking advantage of a data-centric approach.

Since provenance data is so beneficial, we consider that when a user interacts with the workflow execution, new data (user steering data) are generated, and thus their provenance must be registered as well. In a long-running execution, many interactive data analysis and adaptations may occur. If the SWMS does not adequately register the provenance of interaction data, users can easily lose track of what they have changed in the past. This is critical if the entire computational experiment takes days and many adaptations occurred, since it may be impossible to remember in the last day of execution what they have steered in the first days. Furthermore, adding human-adaptation data to the wf-Database enriches its content and enables future interaction analysis. One example of how such data can be exploited is that the registered adaptation data could be used by artificial intelligence algorithms to understand interaction patterns and recommend future adaptations. Thus, the SWMS that enables computational steering should collect provenance of user interaction data. To the best of our knowledge, this has not been done before.

3.2.3 The Role of Scientists in SWMSs

This work aims at supporting one type of scientist, *i.e.*, computational scientists, who are the typical users of a steered workflow. However, running complex data-centric workflows in HPC usually involves several scientists with different levels of expertise on each of the aspects involved in the process. We consider three types of scientists: (a) domain scientist, (b) computational scientist, and (c) computer scientist. All these scientists collaborate to scientific discovery.

(a) Domain scientists. Examples are geologists, biologists, and experimental physicists. They are very familiar with concepts, nomenclature, and semantics of the domain. They are commonly very good at understanding the scientific hypothesis, results and data interpretation. They may not have programming or computational skills. The resulting data of a complex computational simulation are often delivered to them as well organized, cured, and with some aggregations, visualizations, plots, and dashboards. Their main work is typically to give sense to these cured data.

(b) Computational scientists. Examples are engineers, bioinformaticians, and computational physicists. They are not domain specialists, but have knowledge on the domain. However, they are more focused on the computational aspects. They typically have programming and computational skills, and they are familiar with command line interfaces. They are more prone to learning new computing technologies and use new systems that support their computational simulations. They know how to analyze domain-specific data and metadata and organize large raw data files into analyzed data so they can work together with domain scientists to deeply interpret the data. They know how to chain the different simulation programs and design a scientific workflow to attend the main goal of a computer simulation. They are able to operate a SWMS or dispatch jobs in an HPC cluster.

(c) Computer scientists. They are experts in developing tools, methods, or systems that support large-scale simulations. Examples are HPC, data management, workflow solution specialists. They do not necessarily have domain knowledge. Often, computer scientists work closely with computational scientists to obtain the best performance for an HPC simulation and achieve the final goal. They can analyze performance, linked with domain and provenance data to help adjusting the system, debugging, and fixing errors.

4. User-steered Online Data Reduction and Adaptive Monitoring

In this section, we show our main contributions. In the data-centric approach, removing a subset of the entire dataset to be processed means removing a set of input data elements from a dataset to be consumed by a workflow activity. As a consequence of this removal, the tasks that would process the elements within the removed subset will not be executed, hence, reducing both workflow execution time and data processing. Data processing reduction becomes more evident if the removed data elements contain paths to large raw data files that would be consumed by tasks if the elements were not removed. Furthermore, if an input data element of a given activity is removed, the following elements forming the element-flow of the next linked activities will not be processed too, reducing data and, more importantly, execution time in cascade. In Section 4.1, we introduce a new data-centric algebraic operator for user-steered data reduction. In Section 4.2, we discuss the importance of maintaining data consistent after reduction. In Section 4.3, propose an adaptive monitoring approach. In Section 4.4, we explain our approach in the context of existing workflow execution models. In Section 4.5, we exemplify how real-world scientific workflows can highly benefit from our approach.

4.1 An Operator for Data Reduction

Once users analyze data elements that have already been processed, they might identify data elements that will be processed, following the pre-specified dataflow, but will not contribute to the final results. To tackle this, we extend the data-centric algebraic operators (Section 3.1) with a new user-steered operator, Cut, to enable users to cut off a slice containing input data elements.

Definition (Cut): Cut is a user-steered data-centric operator that removes a subset of a dataset R_i based on activity Act that evaluates the criteria C. Its general form is:

$$R_i' \leftarrow Cut(Act, C, R_i)$$

It transforms an input dataset $R_i \in R$ in the dataflow into the output dataset R_i' and the ratio between n input data elements and m output data elements is $n{:}m$, with $n \geq m$. R_i and R_i' follow the same data schema $\boldsymbol{S}(R_i)$. C is the criteria that addresses the slice of input data elements that are removed. C may be either a simple predicate (*e.g.*, $attr_{15} = 'FATIGUE'$) or a minterm predicate (*e.g.*, $attr_{11} > 38 \wedge attr_{12} > 0.1 \wedge attr_{13} < 1.0$).

As any other dataflow operator (*e.g.*, *Map*, *Reduce*, *Filter*, *SplitMap*, etc.), Cut performs data transformation. However, it is not part of the initially designed workflow composition. Differently than the other operators, Cut is not initially defined in the workflow composition. Rather, it is dynamically inserted in the dataflow as a result of a user steering action. Then, after the transformation of R_i into R_i', it is naturally consumed by the subsequent activities that would consume R_i if no reduction happened. Cut allows the SWMS to keep track of user adaptations during a dataset reduction, closely relating the metadata about the human action (*e.g.*, data about the user who performed the action, when the action was performed, how the action occurred, relating with the C criteria used in the reduction, etc.) to the actual data that have been reduced in a Cut. In Section 5.5, we propose extensions to a W3C-PROV compliant data diagram to represent provenance data collected. Figure 5 illustrates how Cut would be dynamically inserted by a user in the Risers Fatigue Analysis workflow, using the algebraic representation.

4.2 Consistency Issues in a User-steered Data Reduction

Cut can only operate on input data elements that are waiting to be processed in the workflow. When a Cut happens, the dataset R_i that will be reduced is a shared resource between the SWMS engine that is normally processing the workflow in a batch job and the user who wants to remove a slice from R_i. Thus, race conditions can occur. Suppose, for example, that at a given instant τ in time, the SWMS finishes processing a set of data elements and then needs to get new data elements that were waiting to be processed. If at the same time τ, the user decides to remove some of those input data elements that were waiting to be processed, the SWMS may go to an inconsistent state because it could try to process elements that were removed. Or, the user may try to remove a slice that the SWMS already considered to process, thus generating errors. These inconsistencies are even more

Initially designed workflow composition	User-steered Cut in a dataset
$RPreprocessing \leftarrow SplitMap\left(\begin{matrix} DataGather, \\ RPreprocessing \end{matrix}\right)$	$RPreprocessing \leftarrow SplitMap\left(\begin{matrix} DataGather, \\ RDataGather \end{matrix}\right)$
$RStressAnalysis \leftarrow Map\left(\begin{matrix} Preprocessing, \\ RPreprocessing \end{matrix}\right)$	$\boldsymbol{RPreprocessing' \leftarrow Cut}\left(\begin{matrix} \boldsymbol{preprocessing\_reduction}, \\ \left\{\begin{matrix} \boldsymbol{wind\_speed < 13} \\ \wedge \\ \boldsymbol{wave\_freq > 1.8} \end{matrix}\right\}, \\ \boldsymbol{RPreprocessing} \end{matrix}\right)$
$RStressCaseSel \leftarrow Filter\left(\begin{matrix} StressCaseSel, \\ RStressAnalysis \end{matrix}\right)$	$RStressAnalysis \leftarrow Map\left(\begin{matrix} Preprocessing, \\ RPreprocessing' \end{matrix}\right)$
$RCFatigueLife \leftarrow Filter\left(\begin{matrix} CurvatureCaseSel, \\ RStressCaseSel \end{matrix}\right)$	$RStressCaseSel \leftarrow Filter\left(\begin{matrix} StressCaseSel, \\ RStressAnalysis \end{matrix}\right)$
$RCompressResults \leftarrow Map\left(\begin{matrix} CalcFatigueLife, \\ RCFatigueLife \end{matrix}\right)$	$RCFatigueLife \leftarrow Filter\left(\begin{matrix} CurvatureCaseSel, \\ RStressCaseSel \end{matrix}\right)$
$RResults \leftarrow Reduce\left(\begin{matrix} CalcFatigueLife, \\ case, \\ RCompressResults \end{matrix}\right)$	$RCompressResults \leftarrow Map\left(\begin{matrix} CalcFatigueLife, \\ RCFatigueLife \end{matrix}\right)$
	$RResults \leftarrow Reduce\left(\begin{matrix} CalcFatigueLife, \\ case, \\ RCompressResults \end{matrix}\right)$

Figure 5. Workflow representation of a user-steered *Cut* operation.

likely to occur in a highly concurrent execution, such as executions on large HPC clusters with thousands of computing cores.

To address this problem, we specify a *safe* subset of an input dataset R_i to be applied the reduction. We split R_i into two subsets P_i and S_i, where P_i is the subset of R_i with input data elements that have already been processed and S_i is the subset of R_i with elements waiting to be processed. Thus, using set theory, $R_i \leftarrow P_i \cup S_i \mid P_i \cap S_i = \emptyset$. S_i is the subset of R_i that is safe to remove a data slice from. To guarantee this, the SWMS must provide lock controls so that only the subset S_i will be reduced. In Section 5.2, we give details about how we implement this in a SWMS.

4.3 Adaptive Monitoring

In this section, we present an adaptive monitoring approach that combines monitoring and human adaptation. It helps users following the evolution of interesting parameters and result data to find which subsets of the dataset can be removed during execution. Also, since what users find interesting may change over time, this approach allows the user to adapt the monitoring definitions, such as which data should be monitored and how. The adaptive monitoring relies on online queries to the continuously populated wf-Database. Users can set up monitoring queries (as in Table 1 and Table 2), analyze monitoring results, and adapt monitoring settings.

Monitoring works as follows. There is a set $\{Q\}$ composed of monitoring queries mq_i, $0 \leq i \leq |\{Q\}|$, each one to be executed at each $d_i > 0$ time intervals. Users do not need to specify queries at the beginning of execution, since they do not know everything they want to monitor. This is why $\{Q\}$ starts empty. After users gain insights from the data, after interactive provenance data analyses, they can add monitoring queries to $\{Q\}$ in an *ad-hoc* manner. Each d_i can be adapted, meaning that users have control of the time frame of each mq_i during execution. The monitoring queries and settings are stored in the wf-Database.

Each mq_i execution generates a monitoring query result set mqr_{it}, $t = \{kd_i \mid k \in \mathbb{N}_{\geq 0}\}$, at each time interval d_i. This result set is also stored in the wf-Database. The users have the flexibility to adapt monitoring during workflow execution. To do so, at each time instant t after each monitoring query result mqr_{it} has been generated, the values for d_i and

mq_i are reloaded from the wf-Database. If any change has happened, it will be considered in the next iteration $t + d_i$. Moreover, at each certain time during execution (also configured by the user), the system checks if the user has added new monitoring queries in $\{Q\}$. Our approach takes full advantage of the data stored online in the wf-Database to enable users to steer monitoring settings, including which data will be monitored and how. We show how an example of a user adapting monitoring settings in Section 7.

4.4 Data Reduction in Workflow Execution Models

Among the different workflow execution models presented in [21], our data reduction can be used in both acyclic (sequential and concurrent) and cyclic models. To exemplify a data reduction in these models, let us consider any two activities Act_i and Act_{i+1} in a given workflow W, where one depends on the other. Using the data-centric algebraic representation, we have:

$$R_{i+1} \leftarrow DT_i(Act_i, R_i);\ R_{i+2} \leftarrow DT_{i+1}(Act_{i+1}, R_{i+1}).$$

In a sequential execution between Act_i and Act_{i+1}, Act_{i+1} only starts to operate when Act_i completely finishes all its tasks [21]. Suppose that the user wants to reduce input data for Act_i. The safe subset S_i contains data elements that are ready to be processed but are only waiting for a free processor, considering that all other processors are still processing tasks from Act_i. This happens when there are more tasks than available processors, which frequently occurs in large-scale workflows. Then, after some tasks for Act_i have been processed, the user can submit analytical queries using the output of Act_i, make a decision, and reduce input data from S_i. In this case, the reduction favors Act_i directly, and Act_{i+1} indirectly because Act_{i+1} has less data to consume. The same occurs when the user wants to reduce input data for Act_{i+1}.

In a concurrent execution between Act_i and Act_{i+1}, there is a pipeline of data elements, *i.e.*, when Act_i finishes processing one task that generates output elements, Act_{i+1} can process those input elements. In this case, the safe subset S_i contains elements that will be processed by Act_i. When they are removed, the tasks from Act_{i+1} that would consume the outputs from Act_i are not executed. In this case, the reduction favors both Act_i and Act_{i+1} directly, since a reduction in Act_i removes a pipeline between Act_i and Act_{i+1}.

Both sequential and concurrent execution models can be iterated in a cyclic model. Thus, at runtime, when the workflow is running in a specific cycle (iteration), online user-steered data reduction can occur. There are four types of cyclic models [2]: (1) counting loops without dependencies between iterations (also known as parameter sweep), (2) counting loops with dependencies (iteration $k + 1$ depends on iteration k), (3) conditional loops ($while\ \ldots\ do$), and (4) dynamic loops (user adapts loop stop condition). In addition to reducing data inside an iteration, as described for the acyclic models, one can reduce iterations using a dynamic loop model [2]. Therefore, our user-steered data reduction approach can be used in almost all the execution models presented in [21], and is a complementary approach to user-steered iteration reduction in cyclic execution models, as done in [2]. We carry out our experimental validation using the case study of Section 2, which combines acyclic (with both concurrent and sequential activities) and cyclic (more specifically, parameter sweep) models.

4.5 Data Reduction in Real-world Scientific Workflows

Several real-world scientific workflows have this same behavior: data elements organized in datasets flowing in a dataflow. To exemplify data reduction in other different workflow execution models, we use (i) SciPhy [10], which iterates over a time consuming input dataset from the bioinformatics domain; and (ii) Montage [11], which has been used to benchmark scientific workflow solutions [22] and represents data-intensive workflows.

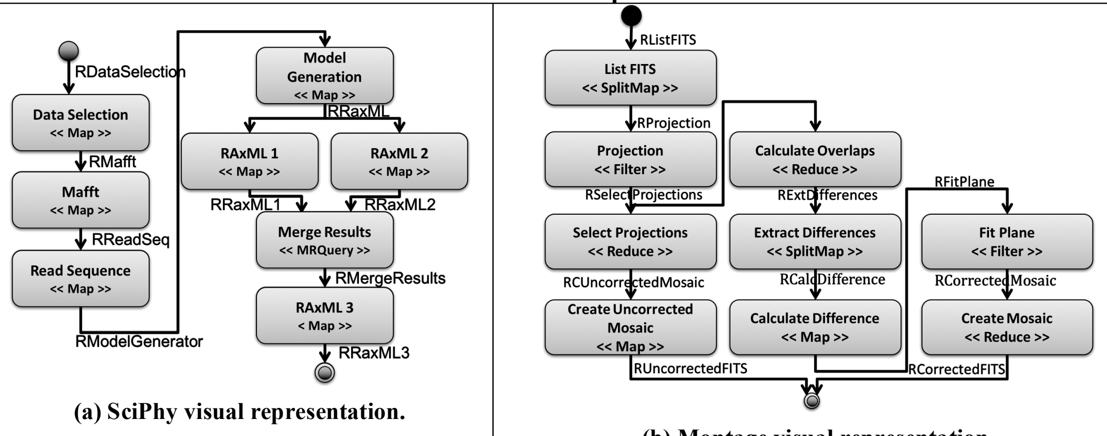
(i) SciPhy [10] is a bioinformatics workflow composed of eight activities for phylogenetic analysis. In Figure 6(a) and Figure 6(c), we show SciPhy with its dataflow and algebraic representation, respectively. SciPhy aims at producing phylogenetic trees that represent evolutionary relationships. These trees are analyzed to identify or discover drug targets. Given an input set of new genomic sequences or genes, specific programs, which are both compute- and data-intensive, are used in a workflow to infer similarity and homology. These sequences are transformed through the dataflow until they arrive at the `Model Generator` activity, which is mostly compute-intensive, and takes a long time to calculate the similarity. Based on previous execution information stored in a provenance database, combined with domain-specific knowledge, the user can tell that a specific combination of genomic sequences will likely take an undesirable amount of time to complete. This is critical for executions in cloud environments, because it will significantly increase the costs. The constraint in this case is cost and not whether the input will lead to interesting results. The user simply cannot pay for the time the program will take to complete a specific slice of the input or the user prefers to spend more time on sequences that will take shorter time and will return results faster, contributing for the overall analysis. Without our solution, the user needs to stop execution, remove this undesired set of genomic sequences by hand, and restart. This is error-prone, the interactions are not integrated with the workflow execution, and it is time consuming.

(ii) Montage [11] is a well-known toolkit for assembling astronomical images into custom mosaics of the sky. The workflow has been modeled using the data-centric algebra [7], with nine activities. In Figure 6(b), we illustrate with a visual representation of the workflow, showing the datasets between each workflow activity. In Figure 6(d), we show how we model Montage using the data-centric algebra [7]. The first activity extracts many Flexible Image Transport System (FITS) files. The contents of these files are data about common astronomy coordinate systems, arbitrary image sizes, rotations, and world coordinate system map projects. Each file has twenty different types of data, modeled as attributes of a dataset in the data-centric approach. The activity `Create Mosaic` builds a mosaic of a delimited region in the outer space and generates an image. Analyzing the generated mosaic, users can infer the presence of an interesting celestial object. Identifying them is hard, subjective, and requires domain expertise.

By correlating specific combinations of input data values captured in those FITS files with the generated mosaics, the domain specialist can tell whether a certain region of the outer space is more or less likely to contain an interesting celestial object. Identifying this correlation is tricky and it may dynamically change during execution, depending on how the data values are evolving. In particular, users investigate the color intensity to detect potential regions of interest that may represent a celestial object. Thus, a pixel with high color intensity may represent an object emitting light or reflecting it. For this reason, the user needs to monitor the results looking for those color intensity changes. Based on the

(i) Montage	(ii) SciPhy

Workflow Visual Representation



(a) SciPhy visual representation.

(b) Montage visual representation.

Data-Centric Workflow Algebraic Representation

$RMafft \leftarrow Map(DataSelection, \quad RDataSelection)$

$RReadSeq \leftarrow Map(Mafft, \quad RMafft)$

$RModelGenerator \leftarrow Map(ReadSeq, \quad RReadSeq)$

$RRaxML \leftarrow Map(ModelGen., RModelGenerator)$

$RRaxML1 \leftarrow Map(RaxML\_1, \quad RRaxML)$

$RRaxML2 \leftarrow Map(RaxML\_2, \quad RRaxML)$

$RMergeResults \leftarrow MRQuery(join, \{RRaxML1, RRaxML2\})$

$RRaxML3 \leftarrow Map(RaxML\_3, \quad RMergeResults)$

(c) SciPhy algebraic representation.

$RProjection \leftarrow SplitMap(mImgtbl, \quad RListFITS)$

$RSelectProjections \leftarrow Filter(mProject, \quad RProjection)$

$RCUncorrectedMosaic \leftarrow Reduce(mAdd, MOSAIC_{ID}, RSelectProjections)$

$RUncorrectedFITS \leftarrow Map(mJPEG, RCUncorrectedMosaic)$

$RExtDifferences \leftarrow Reduce(mOverlaps, MOSAIC_{ID}, RProjection)$

$RCalcDifference \leftarrow SplitMap(extractDiffs, \quad RExtDifferences)$

$RFitPlane \leftarrow Map(mDiff, \quad RCalcDifference)$

$RCorrectedMosaic \leftarrow Filter(mFitPlane, RFitPlane)$

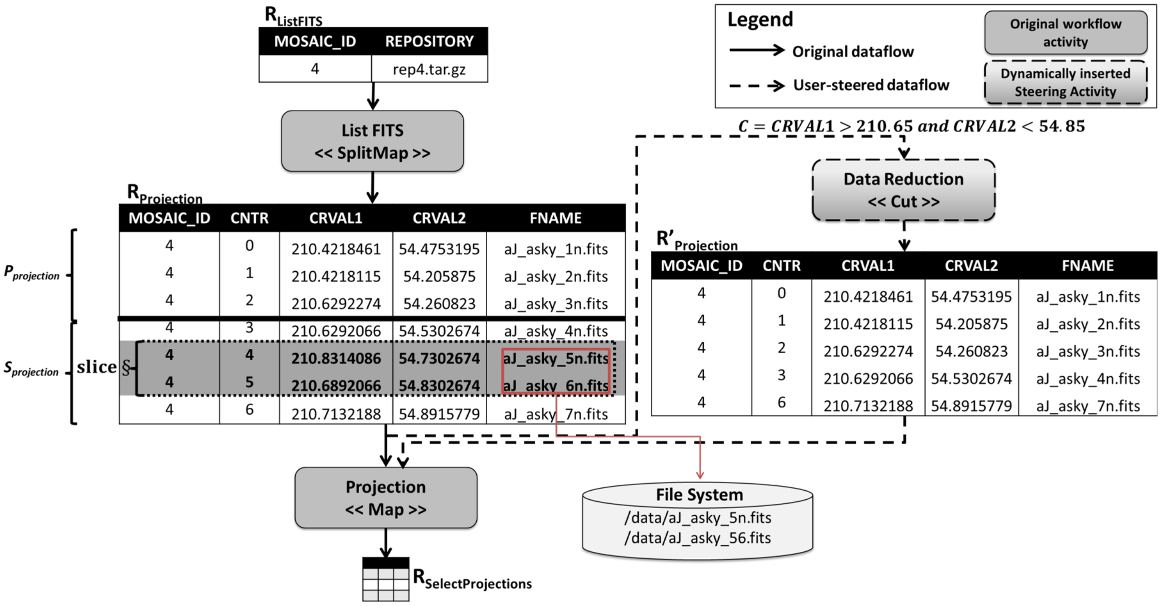
$RCorrectedFITS \leftarrow Reduce(createMosaic, \quad RCorrectedMosaic)$

d) Montage algebraic representation.

Figure 6. Real-world scientific workflows modeled using the data-centric workflow algebra.

results, the user can identify regions of the space that are very unlikely to lead to significant color change. These regions, which are delimited in data values in the input FITS files, could be eliminated at runtime, thus reducing processed data and execution time.

To show a specific case of data reduction, Figure 7 shows an excerpt of the Montage workflow. `List FITS` activity consumes a list of compressed files and each produces a list of many FITS files (*i.e.*, it has a *SplitMap* behavior). The list of FITS files is represented as $R_{projection}$ dataset, which contains paths to the actual files stored on disk. The modeled workflow extracts data values from those files and store in $R_{projection}$ dataset. Among these data values extracted, there are *CRVAL1* and *CRVAL2* that represent two coordinate values to determine a position in the native image coordinate system. The files in $R_{projection}$ are then processed by `Projection` activity, generating $R_{SelectionProjections}$ dataset, following the remainder of the dataflow. After some time has elapsed and files have been processed, a user runs several data analyses in the dataflow and determines that certain values for *CRVAL1* and *CRVAL2* will very unlikely lead to an interesting celestial object identification and these files containing the values can be cut off. Thus, the user does a reduction command using criteria C to cut a slice from $R_{projection}$ dataset, transforming it into a reduced $R'_{projection}$ to be consumed by `Projection` activity in the remainder of the dataflow.



$R_{ListFITS}$

MOSAIC_ID	REPOSITORY
4	rep4.tar.gz

$R_{Projection}$

MOSAIC_ID	CNTR	CRVAL1	CRVAL2	FNAME
4	0	210.4218461	54.4753195	aJ_asky_1n.fits
4	1	210.4218115	54.205875	aJ_asky_2n.fits
4	2	210.6292274	54.260823	aJ_asky_3n.fits
4	3	210.6292066	54.5302674	aJ_asky_4n.fits
4	**4**	**210.8314086**	**54.7302674**	**aJ_asky_5n.fits**
4	**5**	**210.6892066**	**54.8302674**	**aJ_asky_6n.fits**
4	6	210.7132188	54.8915779	aJ_asky_7n.fits

$R'_{Projection}$

MOSAIC_ID	CNTR	CRVAL1	CRVAL2	FNAME
4	0	210.4218461	54.4753195	aJ_asky_1n.fits
4	1	210.4218115	54.205875	aJ_asky_2n.fits
4	2	210.6292274	54.260823	aJ_asky_3n.fits
4	3	210.6292066	54.5302674	aJ_asky_4n.fits
4	6	210.7132188	54.8915779	aJ_asky_7n.fits

Figure 7. User-steered *Cut* in an excerpt of Montage workflow. The dataset $R_{projection}$ is divided into subsets $P_{projection}$ and $S_{projection}$. A slice is cut off from $R_{projection}$ transforming it into $R'_{projection}$ using criteria C.

5. Implementation in d-Chiron

d-Chiron [8] is a SWMS that implements the data-centric approach described in Section 3.1. It collects and stores data at a fine-grain level in the wf-Database during workflow execution. d-Chiron takes advantage of a distributed in-memory database system (MySQL Cluster) to manage the wf-Database. Documentation on how to run d-Chiron and the data schema used to implement the wf-Database are publicly available on GitHub [23]. In this section, we explain how we implement user-steered data reduction and adaptive monitoring in the data-centric approach in d-Chiron.

5.1 Using a Relational DBMS to Implement the Data-centric Approach

Distributed and parallel relational database technology has been successful at managing very large datasets [24]. d-Chiron exploits this technology to support many user steering aspects described in Section 3.2. In this section, we give three reasons to explain why relational DBMS is a good choice to implement data reduction in a data-centric SWMS.

(i) The first motivation is related to the need to find the slice to be removed. A relational DBMS has efficient querying capabilities to enable analysis of sets of data with a query language (SQL) and a query interface. Also, an integrated data modeling using a PROV-compliant data diagram enables complex data queries that analyze scientific domain, provenance, and workflow execution data. This highly contributes to the online analytical capabilities of the SWMS. In Section 6, we show how d-Chiron takes advantage of a query interface to enable users to query the data online to support data reduction.

(ii) The second motivation is related to the fact that scientific workflows are data-centric. There is a flow of data elements organized in sets (datasets). The relational model is set-oriented and there are multiple native constructs that significantly ease managing the flow of data elements. Thus, not only the user can benefit from querying the data, but also the SWMS engine itself. d-Chiron engine uses SQL to access and modify the data in the wf-Database and uses these data in its internals functioning, such as task scheduling. Also, a relational data model eases data reduction. The user does not need to know about the task scheduling details in a data reduction action. Rather, only the slice criteria (domain data only) can be specified and d-Chiron will use SQL to join domain data with scheduling data to select the affected tasks by a reduction. We explain this in details in this section.

(iii) The third motivation is related to consistency. It is essential that the data remains consistent within a user-steered data reduction. It is quite complex to guarantee a consistent execution when a user decides to reduce data, in particular in a large HPC execution and without stopping the workflow execution. On the other hand, all parallel and distributed relational DBMS natively provide atomicity, consistency, isolation, and durability (ACID) transactions [24]. We can take advantage of this capability and implement a user-steered data reduction in a way to outsource to the DBMS complex transaction control that guarantees consistency. We also explain this in this section.

5.2 Supporting Consistent Data Reduction with a Relational DBMS

Before actually reducing the data, we explain how a slice is defined in d-Chiron, using its relational DBMS, so it can later be safely removed while guarantying consistency after reduction. Input data elements are consumed by the many parallel tasks (usually thousands in Many Task Computing workflows [1]) that need to be scheduled by the SWMS engine. Since there is this strong relationship between tasks and domain data elements, which contain scientific domain-specific data values, we represent it in a relational data schema (using crow's foot database notation) where each task is related to one or more data elements of a domain-data dataset:

Task
Domain Dataset
.

All datasets in set R are specializations of $Domain\ Dataset$ and the scheduling of input data elements to parallel tasks is represented as instances in $Task$ table, which stores all tasks in the workflow execution. Thus, for a certain input dataset $R_i \in R$, the join ($R_i \bowtie Task$) returns a set containing tasks with their input data elements in R_i. Moreover, among other attributes, each task has an important $state$ attribute that determines if a task is `READY` to be executed (already knows its input data to start, but is waiting for a free CPU so it can be scheduled), `RUNNING`, `COMPLETED` (already been successfully executed), `BLOCKED` (even though there may be free CPUs, the task does not have the input to start yet), or any other state a task may assume. Depending on the data transformation performed by an activity, each task may consume one (*e.g.*, *Map*, *SplitMap*, *Filter*) or more (*e.g.*, *Reduce*, *MRQuery*) input data elements. Therefore, we distinguish between (i) activities in which each task consumes one data element (we denote such tasks as $tasks^{1:1}$), and (ii) activities in which each task consumes more than one data element (we denote them as $tasks^{1:n}$).

(i) In activities with $tasks^{1:1}$, removing input data element means "informing" the SWMS not to execute the tasks that would consume them, hence reducing overall execution time. To implement the set S_i (Section 4.2), a semi-join relational operation [24] is used to join input data elements from the input dataset R_i with tasks in `READY` state in order to only select the domain data elements that still need to be processed. Then, after having S_i, the SWMS can obtain the elements in S_i that follow the criteria C. Since a set containing both the input data elements together with the related tasks that will consume them is important for the implementation of data reduction, we denote this set as $\S^{1:1}$, so that:

$$\S^{1:1} \leftarrow \sigma_C(R_i) \ltimes \sigma_{state=READY}(Task),$$

where the ratio $1:1$ means that the tasks in this set are $tasks^{1:1}$ (as tasks for Map, Filter, and SplitMap) and the criteria C is defined in the Cut operator. Finally, the SWMS will know that tasks in $\S^{1:1}$ should not be processed.

(ii) In activities with $tasks^{1:n}$, a data reduction in an input dataset R_i can only occur if the task that will consume them is in a `BLOCKED` state. The task has not started yet because the needed input data for it to start is still being generated by a running task in a previous activity. When this running task finishes, it signals that the `BLOCKED` task can start. While it is still blocked and the input data elements are being generated, the user can analyze them and identify data values that can be removed. In this case, we denote the set $\S^{1:n}$ similarly to the previous one, but it rather returns the input data elements in R_i that are being consumed by the tasks in `BLOCKED` state:

$$\S^{1:n} \leftarrow \sigma_C(R_i) \ltimes \sigma_{state=BLOCKED}(Task),$$

where the ratio $1:n$ means that the tasks in this set are $tasks^{1:n}$ (as tasks for Reduce and MRQuery data transformations). Finally, the SWMS will know that tasks in $\S^{1:n}$ should not be processed. In other words, d-Chiron will normally execute the tasks, but with a reduced dataset instead. This is different for $tasks^{1:1}$ because they cannot be executed if their input datasets are removed. The SWMS will know how to handle the tasks in sets $\S^{1:1}$or $\S^{1:n}$ as long as it knows the type of data transformation of the activity that would consume the elements defined by the criteria C. Such verifications are important to guarantee consistency during reduction, which is better explained next.

5.3 Steer Module: User-steered Data Reduction Implementation

To ease slice removal in d-Chiron, we developed the `Steer` module. With the `Steer` module, users can issue command lines to inform the name of the input dataset R_i and the criteria C (see Cut definition). More specifically, the slice delimited by C is added to the *where* clause in the SQL query that will form the *select* expressions. As an implementation decision, instead of physically removing the input data elements (either in $\S^{1:1}$ or $\S^{1:n}$) from the wf-Database, we move them to a `Modified_Elements` table, maintaining the relationships. Likewise, the tasks in $\S^{1:1}$, which cannot be executed, are not physically removed, but they have their state marked as `REMOVED_BY_USER`. By doing so, we enable these tasks and data elements to be later analyzed with provenance queries, similar to the ones shown in Table 1 and Table 2.

To guarantee consistency, we take advantage of d-Chiron's DBMS with ACID transactions. In a user-steered data reduction, both d-Chiron's engine and the `Steer` module need to concurrently update shared resources: `Task` and `Domain Dataset` tables in the wf-Database. The `Steer` module knows if it is about to reduce data elements within a slice of the type $\S^{1:1}$ or type $\S^{1:n}$, since it depends on the dataset being reduced, which is a parameter to the module. With respect to the input data elements (either in $\S^{1:1}$ or in $\S^{1:n}$), while d-Chiron's engine gets the input data elements to execute, the `Steer` module needs to concurrently move the cut off input data elements to the `Modified_Elements` table. With respect to the tasks in $\S^{1:1}$, the `Task` table is a shared resource because while d-Chiron's engine updates the runnable tasks (select them, update their status to `RUNNING`, execute them, and mark them as completed), `Steer` needs to update the `Task` table to mark the tasks as removed by user, so that the engine will not get

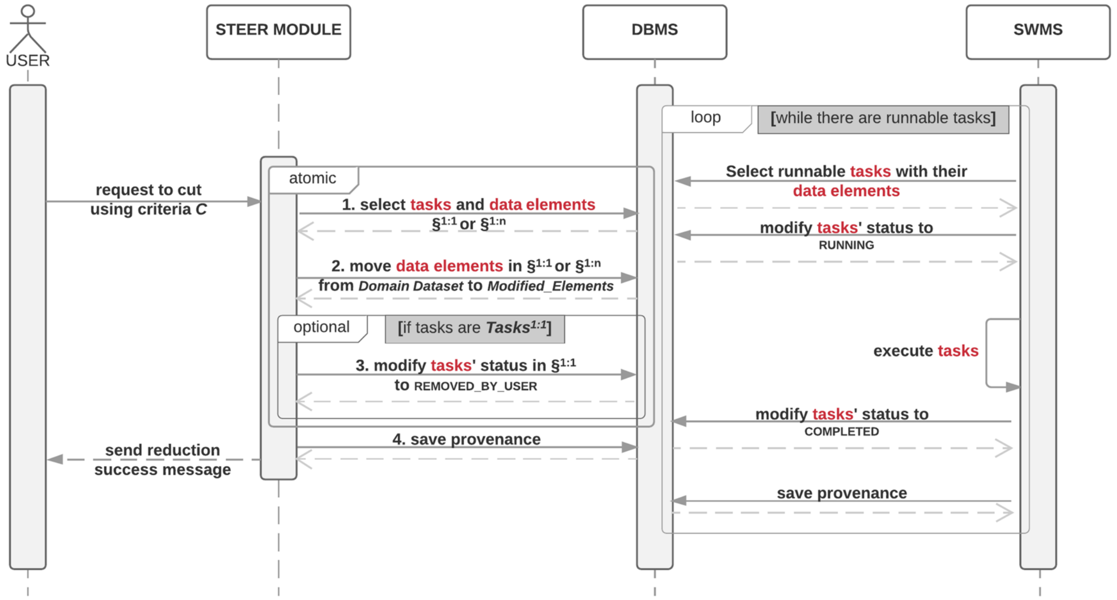

USER
STEER MODULE
DBMS
SWMS
loop
[while there are runnable tasks]
request to cut using criteria *C*
atomic
1. select tasks and data elements
§$^{1:1}$ or §$^{1:n}$
2. move data elements in §$^{1:1}$ or §$^{1:n}$ from *Domain Dataset* to *Modified_Elements*
optional
[if tasks are *Tasks*$^{1:1}$]
3. modify tasks' status in §$^{1:1}$ to REMOVED_BY_USER
4. save provenance
send reduction success message
Select runnable tasks with their data elements
modify tasks' status to RUNNING
execute tasks
modify tasks' status to COMPLETED
save provenance

Figure 8. Sequence diagram showing what happens in a user-steered data reduction.

them for execution. These concurrent actions make concurrency control critical. Figure 8 illustrates these steps with a sequence diagram. The `Steer` module acts concurrently with the SWMS engine on the shared resources, which are in red in the figure. The steps 1-3 in `Steer` module are put together in a single DBMS transaction, which is atomic.

The wf-Database' tables are distributed, thus making concurrency control of the tables' partitions even more complex. In d-Chiron engine, distributed concurrency control in these tables is outsourced to the DBMS that guarantees the ACID properties [24]. We developed the `Steer` module to also exploit the DBMS in a way that the concurrency caused by the aforementioned updates is controlled by the DBMS. Therefore, we implement our approach such that both d-Chiron engine and the `Steer` module rely on the DBMS to outsource those complex distributed locks and releases of shared resources to guarantee that both execution and data remain consistent before and after a user-steered reduction.

To store provenance of removed data elements, we extend the wf-Database schema with the table `User_Query` to store the queries that select the slice of the dataset to be removed. The description for each `User_Query` column is described in Table 3. We keep track of the removed data elements in table `Modified_Elements`, which is a table that represents a many-to-many relationship between `User_Query` and `Domain Dataset`.

Table 3. `User_Query` **table description.**

Column name	Description
`query_id`	Auto increment identifier
`slice_query`	Query that selects the slice of the dataset to be removed.
`tasks_query`	Query generated by the SWMS to retrieve the ready tasks associated.
`issued_time`	Timestamp of the user interaction
`query_type`	Field that determines how the user interacted. It could be "Removal", "Addition", and others.
`user_id`	Relationship with the user who issued the interaction query
`wkfid`	To maintain relationship with the rest of workflow execution data.

5.4 *Monitor Module: Adaptive Monitoring Implementation*

To implement our approach to adaptive monitoring, we extend the wf-Database schema. To store $\{Q\}$, we add the table `Monitoring_Query`, shown in Table 4. The main advantage of storing monitoring results in the wf-Database (and adequately linking the results with the remainder of the data already stored in this database) whenever a monitoring query result is executed is that users are able to query the results immediately after their generation. The wf-Database can also serve as data source for data visualization applications. We add another table: `Monitoring_Query_Result`, shown in Table 5, to store monitoring results in the wf-Database. In Section 5.5, we show the extensions of PROV-Wf for these adaptive monitoring concepts.

Table 4. `Monitoring_Query` **table description.**

Column name	Description
`monitoring_id`	Auto increment identifier
`interval`	Interval time (in seconds) between each monitoring query (d_i)
`monitoring_query`	Raw SQL query to be executed.
`wkfid`	Relationship between the monitoring queries and the current execution of the workflow. In d-Chiron's wf-Database, there may be data from past executions for a same workflow.

Table 5. `Monitoring_Query_Result` **table description.**

Column name	Description
`monitoring_result_id`	Auto increment identifier
`monitoring_id`	Relationship with the monitoring query that generated this result
`monitoring_values`	Results of the `monitoring_query`
`result_type`	Data type of the result values of both queries. Currently, "Integer", "Double", "Array[Integer]", and "Array[Double]"

A command line starts the `Monitor` module that runs in background. It establishes a connection with the distributed DBMS. Connection settings are provided in a configuration file. d-Chiron makes use of this configuration file to define the workflow design, workflow general settings, and other user-defined variables. Then, the `Monitor` program keeps querying the `Monitoring_Query` table at each s time units to check if a new monitoring query was added. The default value for s is 30 seconds, as the time interval to check if monitoring queries were added or removed. Users can customize this value. After the `Monitor` has started, users can add (or remove) monitoring queries to (or from) the `Monitoring_Query` table. Currently, users can add monitoring queries using a command line to inform which SQL query will be executed at each time interval and the time interval. Whenever the `Monitor` module identifies that the user added a new monitoring query, it launches a new thread. Each thread is responsible for executing each monitoring query in

```
1.  Execute the monitoring query $\boldsymbol{mq_i}$
2.  Store query results in the wf-Database
3.  Reload all information for $\boldsymbol{mq_i}$ from the wf-Database for the next time iteration. The user could
    have adapted any of this information.
4.  Wait for $\boldsymbol{d_i}$ seconds
```

Figure 9. Steps executed by each thread within a time interval.

Monitoring_Query at each defined time interval. A thread is finished when a monitoring query is removed or when the workflow stops executing (in that case, all threads are finished). Figure 9 shows the steps executed at each time interval.

To enable all these steering capabilities, three of these steps represent queries to the wf-Database, including reads and writes. The stored results can be further analyzed *a-posteriori* or, more interestingly, used as input for runtime data visualization tools, since results are immediately made available after they are generated. In Section 6, we show how users can interact with d-Chiron to add monitoring queries and use the Steer module.

5.5 Extending PROV-Wf for Data Reduction and Adaptive Monitoring

The data schema that governs the data organization in the wf-Database follows PROV-Wf [6]. It adheres the W3C PROV [25] recommendations to help query specification, to maintain compatibility between different SWMSs, and facilitate interoperability between different databases. PROV-Wf specializes PROV concepts for scientific workflows. It is called PROV-Wf, which allows for domain, execution, and workflow provenance data representation at a finer grain than PROV.

We propose extensions to PROV-Wf to accommodate the concepts presented in this work. These concepts extended are: UserQuery, MonitoringQuery, and MonitoringResult, as in Figure 10. Using PROV nomenclature, UserQuery is a *PROV Activity* that stores the slice that represents sets of data elements that will be removed. MonitoringQuery is a *PROV Activity* that contains the monitoring queries submitted by the user in specific time intervals. The monitoring queries generate *PROV Entity* MonitoringResult that stores the query results.

6. Using d-Chiron

The ultimate goal of this work is to contribute with user-steered workflows in HPC. In Section 3.2, we explained that there are at least six aspects that need to be considered for this: interactive analysis, monitoring, human adaptation, notification, interface for interaction, and computing model [4]. In this work, we mostly focused on the first three, considering human adaptation as the core of computational steering and the one we mostly

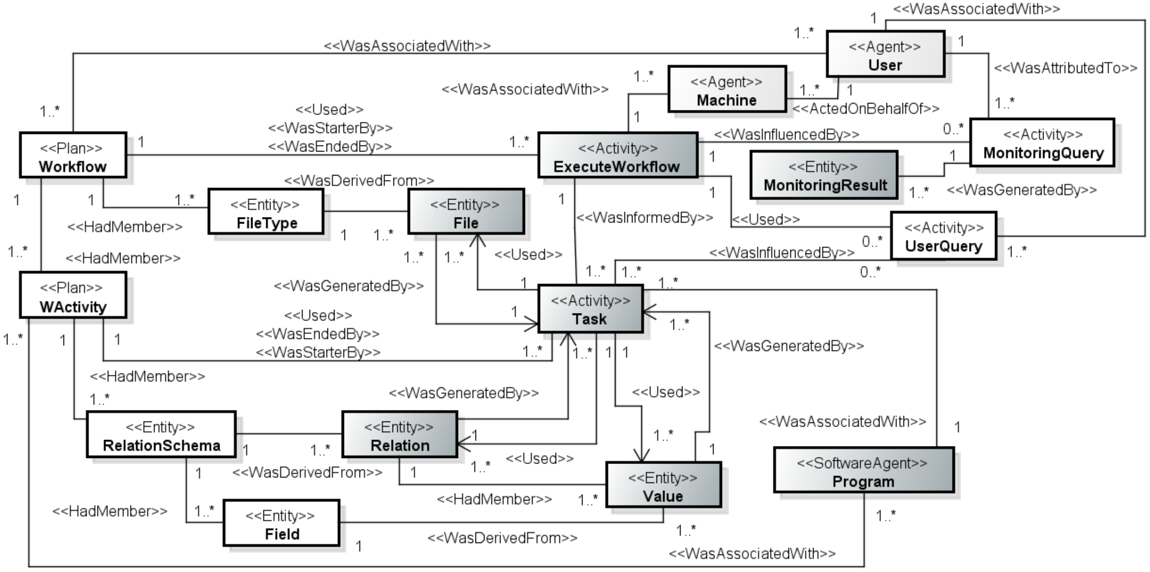


Figure 10. Extended PROV-Wf entity-relationship diagram to accommodate modified tasks and monitoring.

contributed with. As most related contributions to putting the human in the loop of HPC workflows [2,26,27], we focused on the efforts for engineering the backend enabling technology for user steering in an HPC workflow. More specifically, we contributed with allowing users to steer data reductions in scientific workflows online, focusing on providing a consistent execution within a data reduction, managing provenance data of user steering actions, and minimizing performance overheads in the HPC system (discussed in Section 7.4). Enabling such features without jeopardizing performance in an HPC environment is sufficiently complex. However, besides engineering the backend enabling technology, the interface for interaction is another aspect to be considered.

Designing good interfaces requires usability studies to determine whether the interfaces are in fact good for the target user profile (*i.e.*, computational scientists in our case). This would need a comprehensive user experience test to understand user behavior while interacting with their workflows, then we would develop interfaces based on the gathered design insights, and evaluate the usability. For a valid and comprehensive usability evaluation, we would need to ask multiple users to use the system and the modules developed, observe how they use, and interview them. However, the general scenario (HPC workflows) is very complex. Our user profile is quite rare (compared with general business applications) and the results depend on the domain and on the application in the domain. For example, if we want to measure the time a user takes to identify that a certain slice will not contribute for the final results and then remove it, a valid evaluation would require analyzing multiple users of a same application, in a same domain. Finding users of a same specific application is so rare that makes a comprehensive usability test extremely hard. Also, many other questions need to be addressed. For instance, "does the user expertise in the domain-application interfere in the results? – perhaps the more experienced the user is, the faster she will find which slice to remove and the better she understands the consequences of a reduction"; or "what if the tests were carried out on a different application for a same domain?"; or "what about a different domain?". Additionally, an extra raises because using an HPC cluster requires scheduling. For a best usability test, the analyzer needs to observe the user while she is interacting with the HPC workflow, and thus the analyzer's and the user's scheduling must match the HPC job scheduling time, for each user. In other words, a valid and comprehensive usability test would require observing users of a same application, of different domains, of different expertise levels, and matching scheduling times with the HPC cluster. Combining these requirements makes it very hard and out of the scope of this work, which focuses on the enabling backend technology for steering an HPC workflow.

Therefore, instead of usability tests, in this section, we show how users can use our modules `Steer` and `Monitor` in d-Chiron. Before developing, we interviewed few computational scientists. We found that they are very used to command line interfaces and they frequently have to learn new computational tools. They often browse logs in terminals and follow execution status of their simulations. To reduce data, they need to stop their workflow process, modify the input datasets by hand, and restart execution. For some users, this means resubmitting a job to an HPC cluster subject to scheduling. This may be really long (even weeks). Therefore, developing a technology that allows them to reduce data online, based on provenance data analysis through structured queries (rather than Unix-like shell commands to filter multiple logs in the file system – which is what most of them do) is a very desirable feature and we are not aware of any other SWMS that provides it. Thus, we developed simple command line interfaces to enable them to steer monitoring queries and reduce data online. Since developing the best interface and analyzing its usability is out of the scope, the command line interfaces are our current best effort to make the technology usable. As we show in Section 7, for validation purposes, a real user is able to use the system, after a d-Chiron specialist trained her and provided support.

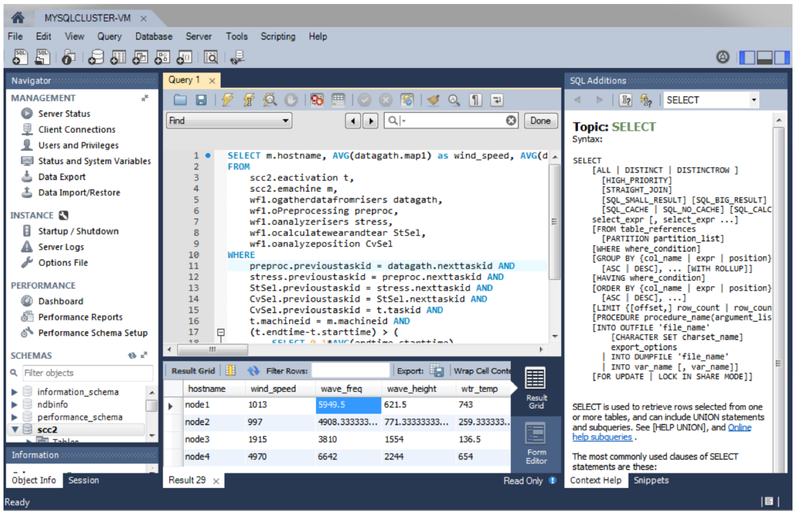
In d-Chiron, computer scientists who are experts in operating d-Chiron work closely with computational scientists, the target-user of this work. However, computational scientists are able to operate d-Chiron and steer the running workflow. Before steering a workflow, the workflow has to be modeled. The user identifies the input and output data elements of each of those activities and gives a name to the dataset that contains those elements (following the data-centric algebra presented in Section 3.1). When this is done, d-Chiron modules creates tables corresponding to those datasets, where each table column is an element produced or consumed by a workflow activity. If needed, application-specific extractor scripts are built to collect output data to be stored in the wf-Database [7].

The aspects of computational steering workflows tackled in this work are strongly related to how d-Chiron manages the data and the dataflow. It is all about the wf-Database being populated online by the SWMS. The workflow execution plan depends on the data in this database (hence can be adapted at runtime) and the wf-Database is available for user queries immediately after the workflow has started to run and data elements in the domain-dataflow are stored while they are generated. Then, they are linked to execution and provenance data in the wf-Database to enable queries that integrate all these data.

Therefore, the main way users can interact with a workflow execution in d-Chiron is through query interfaces, generally provided by the DBMS, to query the wf-Database. To be able to run queries, the user must understand the database schema [23] that logically organizes data in d-Chiron. Computational scientists can work with computer scientists (d-Chiron experts in this case) so they can build complex analytical SQL queries to interactively analyze the dataflow. From our experience, computational scientists do not take much time to learn how to write simple queries to a relational database and they later learn how to write complex analytical queries on their own. d-Chiron uses MySQL Cluster to manage its wf-Database. MySQL users are accustomed to using MySQL Workbench as a visual interface to the DBMS. They can see the relational database schema, build and run SQL queries, and get their tabular results in the interface as the workflow runs. Figure 11 shows how MySQL Workbench can be used to write Q5 (from Table 2 described as natural language in Section 3.2.1), which integrates domain, provenance and execution data in a same query. More queries with their natural langue descriptions are on GitHub [23].

However, such interactivity does not need to use SQL queries only. Some users prefer graphical user interfaces, so there are many other ways to interact with a DBMS: graphical interfaces with drag and drop boxes to help building queries, Natural Language to Database solutions to translate regular English sentences into SQL queries, dashboards that plot results from a query, etc. d-Chiron developers have been working on new tools to facilitate such interactions, including dashboards that plot monitoring charts or command line tools that does not require users to type raw SQL but simpler commands. In this work, discussing the usability of these interfaces is not the focus. However, we show how users currently use command line interfaces for the modules `Steer` and `Monitor`.

For the `Steer` module (Figure 12), a user informs who is going to interact (this



```sql
SELECT m.hostname, AVG(datagath.map1) as wind_speed, AVG(d
FROM
    scc2.eactivation t,
    scc2.emachine m,
    wf1.ogatherdatafromrisers datagath,
    wf1.oPreprocessing preproc,
    wf1.oanalyzerisers stress,
    wf1.ocalculatewearandtear StSel,
    wf1.oanalyzeposition CvSel
WHERE
    preproc.previoustaskid = datagath.nexttaskid AND
    stress.previoustaskid = preproc.nexttaskid AND
    StSel.previoustaskid = stress.nexttaskid AND
    CvSel.previoustaskid = StSel.nexttaskid AND
    CvSel.previoustaskid = t.taskid AND
    t.machineid = m.machineid AND
    (t.endtime-t.starttime) > (
        SELECT 0.1*AVG(endtime-starttime)
```

hostname	wind_speed	wave_freq	wave_height	wtr_temp
node1	1013	5949.5	621.5	743
node2	997	4908.333333...	771.33333333...	259.333333...
node3	1915	3810	1554	136.5
node4	4970	6642	2244	654

SELECT

Topic: SELECT
Syntax:

```
SELECT
    [ALL | DISTINCT | DISTINCTROW ]
      [HIGH_PRIORITY]
      [STRAIGHT_JOIN]
      [SQL_SMALL_RESULT] [SQL_BIG_RESULT]
      [SQL_CACHE | SQL_NO_CACHE] [SQL_CALC
    select_expr [, select_expr ...]
    [FROM table_references
      [PARTITION partition_list]
    [WHERE where_condition]
    [GROUP BY {col_name | expr | position}
      [ASC | DESC], ... [WITH ROLLUP]]
    [HAVING where_condition]
    [ORDER BY {col_name | expr | position}
      [ASC | DESC], ...]
    [LIMIT {[offset,] row_count | row_coun
    [PROCEDURE procedure_name(argument_lis
    [INTO OUTFILE 'file_name'
        [CHARACTER SET charset_name]
        export_options
      | INTO DUMPFILE 'file_name'
      | INTO var_name [, var_name]]
    [FOR UPDATE | LOCK IN SHARE MODE]]
```

SELECT is used to retrieve rows selected from one or more tables, and can include UNION statements and subqueries. See [HELP UNION], and Online help subqueries .

The most commonly used clauses of SELECT statements are these:

Figure 11. Using MySQL Workbench to query wf-Database at workflow runtime

information is stored in the wf-Database for provenance) and passes a configuration file that contains information about the workflow, the HPC cluster, and the DBMS connection settings. After that, a user can run as many dataflow steering commands as necessary informing the input dataset R_i in the workflow that will be reduced and the C criteria to select the slice (operands from Cut definition in Section 4.1). These actions are stored in the wf-Database for provenance. The response messages (in green) allow the user to understand what is happening after a command line is issued. In particular, after a Cut action, the response message informs the user of the number of data elements that were removed from the dataset to be processed. For more complex analyses on the consequences of those reductions, users can query the wf-Database using the tables introduced in this work to verify, for example, if there were files and their sizes to quantify the number of bytes that were not processed. We do those analyses in the next section.

```
1. $> ./Steer --user="Peter" --conf=SC.xml
Next workflow interactions will be issued by user Peter.
2. $> ./Steer --cut --dataset="opreprocessing" --criteria="wind_speed < 12.0 and wave_freq
> 2.0"
177 data elements were cut off from OPREPROCESSING dataset.
3. $> ./Steer --cut --dataset="opreprocessing" --criteria="wind_speed < 11.3 and wave_freq
> 1.8"
55 data elements were cut off from OPREPROCESSING dataset.
```

Figure 12. Steer module command line interface.

```
1. $> ./Monitor --start --conf=SC.xml
System is ready to accept new monitoring queries.
2. $> ./Monitor --add --mq="`cat q1.sql`" --label="query 1" --interval=30
Monitoring query "query 1" will be executed at each 30 seconds.
3. $> ./Monitor --add --mq="`cat q2.sql`" --label="query 2" --interval=20
Monitoring query "query 2" will be executed at each 20 seconds.
4. $> ./Monitor --update --label="query 2" --interval=5
Monitoring query "query 2" was updated. It will be executed at each 5 seconds.
5. $> vi q1.sql
6. $> ./Monitor --update --label="query 1" --mq="`cat q1.sql`"
Monitoring query "query 1" was updated.
```

Figure 13. Monitor module command line interface.

For the `Monitor` module (Figure 13), a user runs a command to start the monitoring module as a background process on any cluster node that has access to the DBMS, usually the same node from which the SWMS execution was launched. Then, users can add monitoring queries at any time. The monitoring query results are also properly stored in the wf-Database, as they are generated. Dashboard graphic visualization applications can query these results to deliver better data visualization for the user.

In this example, after the user starts the monitoring module (line 1), two monitoring queries are added with intervals 30 and 20 seconds, respectively (lines 2 and 3). The user wrote the queries in text files (q1.sql and q2.sql), which are loaded in the `Monitor --add` commands, using cat Unix command. Those query files are only to facilitate the command lines and they are not a requirement. A user could write the query string directly in the command line. After some time, user decides to decrease the time interval in the monitoring of query with label "query 2" by issuing line 4. In line 5, user decides to modify a specific query aspect (*e.g.*, increase the result limit) by editing the query text file and in line 6 he modifies the monitoring query. All these interactions are properly stored in the wf-Database for provenance, following the data schema extensions provided in Section 5.5.

7. Experimental Validation

In this section, we validate our solution for online data reduction based on a real case study. Section 7.1 shows the experimental setup. Section 7.2 presents a use case where the user deals with the `Steer` and `Monitor` modules. Section 7.3 provides broader analyses of reduction and Section 7.4 analyzes the added overheads.

7.1 Experimental Setup

Scientific workflow. We use the Riser Fatigue Analysis workflow (see Figure 1), which is based on a real-world case study. The workflow processes over 350 GB of raw data. In all executions, we use the same dataset, which spans over 60,000 data elements to be processed in parallel. Depending on the workflow activity, tasks may take few seconds (*e.g.*, Activity 1) or up to one minute on average (*e.g.*, Activity 3). The execution model is an iterative workflow (parameter sweep) with concurrent activities, except for the last activity (Activity 7), which is a *Reduce* that requires that Activity 6 completely finishes before it can start.

Software. In all executions, we use d-Chiron [8] with MySQL Cluster 7.4.9 as its in-memory distributed database system to manage the wf-Database. The code to run d-Chiron and setup files are available on GitHub [23].

Hardware. The experiments were conducted in Grid5000 using a cluster with 39 nodes, containing 24 cores each (summing 936 cores). Every node has two AMD Opteron 1.7 GHz 12-core processors, 48GB RAM, and 250GB of local disk. All nodes are connected via Gigabit Ethernet and access a shared storage of 10TB.

7.2 Test Case

Let us consider the following scenario. Peter is an offshore engineer, expert in riser analysis and learned how to set up monitoring, analyze d-Chiron's wf-Database, and use the `Steer` module developed in this work. In Peter's project, the Design Fatigue Factor is set to 3 and service life is set to 20 years, meaning that fatigue life must be at least 60 years (Section 2). Peter is only interested in analyzing risers with low fatigue life values, because they are critical and might need repair or replacement. During workflow execution, it would be interesting if Peter could inform the SWMS, which input values would lead to low risk of fatigue, so they could be removed. However, this is not simple because it is hard to determine the specific range of values (*i.e.*, the slice to be removed). For this, Peter first needs to understand the pattern of input values associated to low risk of fatigue life values. In the workflow (Figure 1), the final value of fatigue life is calculated in Activity 6, but input values are obtained as output of Activity 1, gathered from raw input files. Keeping provenance is essential to associate data from Activity 1 with data from Activity 6.

To understand which input values are leading to high fatigue life values, Peter monitors the generated data online. For simplicity, we consider *wind speed*, which is only one out of the many environmental condition parameter values captured by Activity 1 to serve as input for Activity 2. Peter knows that wind speed has a strong correlation with fatigue life in risers. He expects that with low speed winds, there is a lower risk of accident.

When workflow execution starts, the `Monitor` module is initialized. Then, Peter adds two monitoring queries: (mq_1) shows the average of the 10 greatest values of fatigue life calculated in the last 30s of workflow execution, setting $d_1 = 30$ s; and (mq_2) shows the average wind speed associated to the 10 greatest values of fatigue life calculated in the last 30s, also setting the query interval $d_2 = 30$s. We recall from Table 1 that mq_1 is similar to $Q1$, but only considering data processed in the last 30 s. mq_1 and mq_2 queries are added to the `Monitoring_Query` table.

Peter monitors the results using the `Monitoring_Result` table. These results can be a data source for a visualization tool that plots dashboards dynamically, refreshed according to the query intervals. After gaining insights from the results and understanding patterns, he can start removing the undesired values for wind speed. The monitoring query results mqr_{1t} and mqr_{2t} for the two previously listed queries, as well as when the user reduced the data, are plotted along the workflow elapsed time, as shown in Figure 14. It shows mqr_{1t} (*Fatigue life*) in green line with square markers and mqr_{2t} (*Wind speed*) in blue line with triangle markers. These markers determine when the monitoring occurred.

The workflow execution starts at $t = 0$, but only after approximately 150 s, the first output results from Activity 6 start to be generated. From the first results, at *t*=150 and *t*=180, Peter checks that when wind speed is less than 16 km/h (see horizontal dashed line in $wind\ speed = 16$ in Figure 14, the results lead to the largest fatigue life values. Since risers with large fatigue life values are not interesting in this analysis, he decides, at t =190, to remove all input data elements that contain wind speed less than 16 km/h. For this, the first user query q_1 is issued with a command line to the `Steer` module. User queries are represented with red circles in the horizontal axis (*Elapsed time*). The time a user issued an

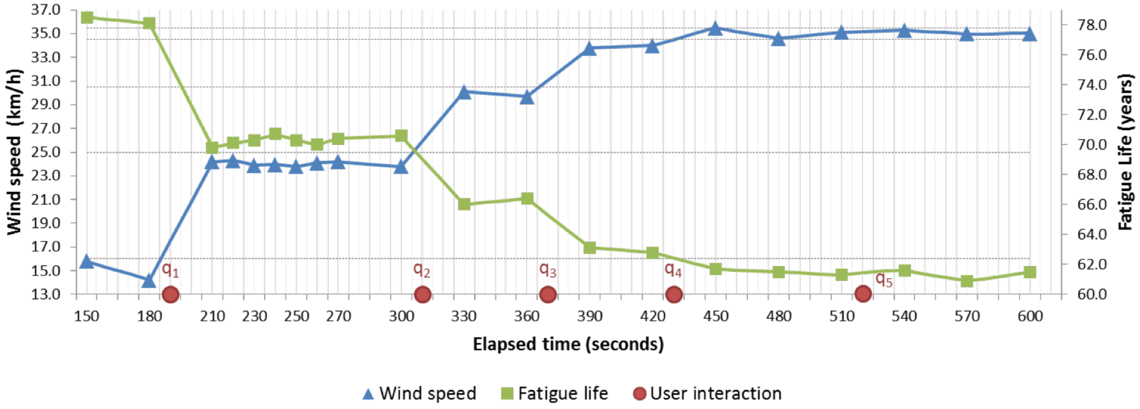


Figure 14. Use case plot to analyze impact of user steering comparing Wind Speed (input) with Fatigue life.

interaction query is stored in User_Query table.

The next marker after q_1 happens at $t = 210$. Comparing with the previous monitoring mark, at $t = 180$, we can observe that this Peter's steering (q_1) increases the minimum wind speed values to be considered from 14.2 km/h to 24.1 km/h. Also, we observe a significant decrease in the slope of the largest values for fatigue life (10.6% lower). This means that the removal of these input data containing wind speed less than 16 km/h made the SWMS not process data containing low wind speed values, which would lead to larger fatigue life results.

Then, monitoring continues, but that slope decrease calls Peter's attention. To obtain a finer detail of what is happening, he decides to adjust monitoring settings, the monitoring interval times (d_1 and d_2) in this case, at runtime. He reduces them to 10 s to get monitoring feedbacks more frequently. We can observe that for both lines mqr_{1t} and mqr_{2t}, the markers become more frequent during $t = [220, 270]$. This is because a monitoring is registered at every 10 s. Although we show monitoring correlations between wind speed and fatigue life, other monitoring correlations could also be analyzed and users can add, remove or adjust monitoring queries at any time during execution. After verifying that the results are reasonable, he decides to adjust the monitoring setting to increase back the monitoring query intervals for both queries to 30s after $t = 270$. Then he observes that since q_1, wind speed less than 25 km/h are leading to large fatigue life values. Then, at $t = 310$, he calls Steer again to issue q_2 that removes input data for wind speed < 25 km/h. The next markers after q_2 shows that this steering made the wind speed value associated to large fatigue life be at least 30.5 Km/h and a decrease of 6.5% in large fatigue life values between $t = 300$ and $t = 330$.

Similarly, Peter continues to monitor and steer the execution. He issues q_3 at $t = 370$ to remove input data with wind speed < 30.5 km/h, making a decrease of 4.9% in large fatigue life (comparing fatigue life in $t = 360$ and $t = 390$). Then, he issues q_4 at $t = 430$ to remove input data with wind speed < 34.5, attaining a decrease of 1.7% in large fatigue life (comparing fatigue life in $t = 420$ and $t = 450$). Despite this small decrease, he decides at t = 520 to further remove data, but with wind speed < 35.5 km/h. However, no decrease greater than 1% in the large fatigue life values was registered after this last Peter's steering. Thus, he keeps analyzing the monitoring results, but does not remove input data anymore until the end of execution.

We store each interaction in the User_Query table and map (in table Modified_Elements) its rows with rows in Domain Dataset and Task tables, to consistently keep provenance of which data elements were modified (in this case, removed) by each specific user steering. Thus, keeping provenance of user steering helps analyzing how specific interactions impacted the results. Figure 14 shows that some specific interactions imply significant changes in lines' slopes (key output values for the user).

7.3 Analyzing User-steered Data Reduction

In this section, we analyze how those previous user interactions impact the amount of resources saved during the workflow execution. More specifically, we analyze three aspects: (i) the number of data elements reduced, (ii) the time that was saved due to the input data not processed, and (iii) the number of bytes of the raw data files that were not processed. For validation purposes, we can count the resources saved as consequences of a data reduction. For this, we compare the executions with and without user-steering. We run the exact same workflow and input datasets for both scenarios. The workflow execution with no steering processes all input data, including those containing wind speed values that lead to risers with low risk of fatigue, which are not valuable for Peter's analyses.

In Figure 15, we depict the three analyzed aspects per activity in the workflow. In other words, we count the total input data elements each activity consumes; the total number of gigabytes of data files processed in each activity; and the total time each activity took to complete. In total, considering all activities, the workflow with no steering processed 60,939 input data elements in parallel, 356GB of domain data files, and the overall execution time was 16.3 min running on the 936-cores cluster.

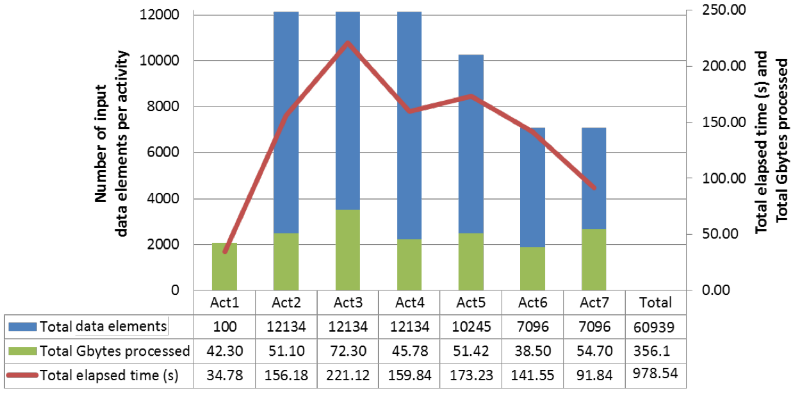
Then, we can compare these numbers with analogous numbers in the scenario with user-steered data reductions. Table 6 summarizes the user interactions (*i.e.*, user-steered reductions) performed as described in the previous section. Figure 16 illustrates how each interaction q_i affected the three analyzed aspects in each workflow activity: Figure 16(a) shows the number of input data elements reduced, Figure 16(b) shows the time saved, and Figure 16(c) shows the amount of gigabytes not processed due to data reduction. In the three charts, although the reductions happen in dataset R_2 consumed by Act_2, we can see that they impact all subsequent activities (Act_1, which is a preceding activity, is not affected by the reductions). In particular, we can see that the first interaction q_1 alone causes a time reduction of 15% (q_1 makes Act_3 complete 33s faster, whereas without reductions Act_3 would take 221 s).

Figure 17 shows the summary of the impacts in the entire workflow by each interaction q_i. Overall, the steering reductions in this experimental validation yield a reduction of 7,854 out of 60,939 data elements (12.89%), including elements in R_2 and elements in subsequent datasets as consequences of the reduction in R_2. Also, the interactions make the SWMS not process 51GB out of 356GB (14.3% of data files processing reduction) and the activities run faster, reducing in total 5.3 min out of 16.3 min (32% of total workflow execution time reduction) in the 936-cores cluster. In particular, we see that the first user-steered reduction q_1 represents 45% of the total amount of time saved, meaning that at the beginning, the user can identify a large slice of the input data that would not lead to interesting results, and we see that the last interaction q_5 did not considerably affect execution. These results were obtained by querying the wf-Database at the end of execution.

7.4 Analyzing the Monitoring Overhead

The SWMS we used to implement our solution implements the data-centric algebraic approach and captures and manages domain dataflow, provenance, and execution data in a fine-grain level during execution, enabling users to query these data online. These functionalities add some overhead. Measuring this overhead is out of the scope of this work, but some measurements are provided in [8,14,28].

However, in this section, we discuss the overhead caused by the solutions proposed in this work. First, when a user-steered data reduction happens, there are data movements in the wf-Database, *i.e.*, some tasks and input data elements are updated or transferred from a table to another (see Section 5.3). Time spent doing these updates in the database is significantly lower than the overall workflow execution time. In fact, each data reduction

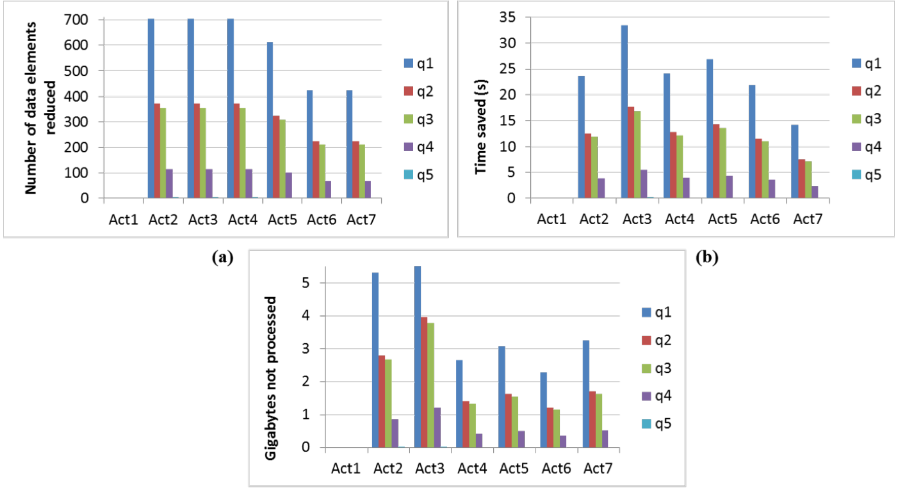


	Act1	Act2	Act3	Act4	Act5	Act6	Act7	Total
🟦 Total data elements	100	12134	12134	12134	10245	7096	7096	60939
🟩 Total Gbytes processed	42.30	51.10	72.30	45.78	51.42	38.50	54.70	356.1
🟥 Total elapsed time (s)	34.78	156.18	221.12	159.84	173.23	141.55	91.84	978.54

Figure 15. Total data elements, gigabytes, and time consumed by workflow activity running with no user steering.

Table 6. Summary of the user-steered reductions (q_1–q_5) with their user-defined slices.

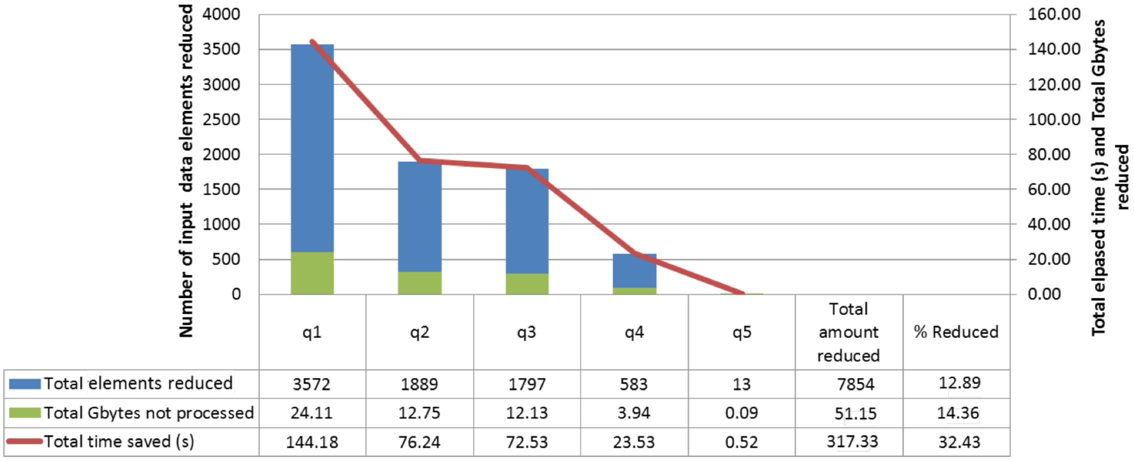
Interaction	Issued time (s)	Slice query
q_1	190	`wind_speed < 16`
q_2	310	`wind_speed < 25`
q_3	370	`wind_speed < 30`
q_4	430	`wind_speed < 34.5`
q_5	520	`wind_speed < 35.5`



(a) **(b)**

(c)

Figure 16. Reduced resources by activity caused by each user-steered reduction q_i.



	q1	q2	q3	q4	q5	Total amount reduced	% Reduced
🟦 Total elements reduced	3572	1889	1797	583	13	7854	12.89
🟩 Total Gbytes not processed	24.11	12.75	12.13	3.94	0.09	51.15	14.36
🟥 Total time saved (s)	144.18	76.24	72.53	23.53	0.52	317.33	32.43

Figure 17. Summary of all online user-steered reductions (q_1–q_5) in the workflow.

q_i (Table 6) takes less than 1 second to finish, whereas the overall execution time of the workflow, after the reductions, is 661 s. Thus, we consider those data movements' overhead negligible. Second, our adaptive monitoring solution adds overheads and need to be measured. Recall that every monitoring query mq_i in $\{Q\}$ is run by a thread at each d_i seconds. Depending on the number of threads ($|\{Q\}|$) and on the interval d_i there may be too many concurrent accesses to the wf-Database, which may add overhead.

To measure this, we set up the `Monitor` module to run queries, which are variations of the queries $Q1$-$Q7$ (Table 1 and Table 2). For example, in $Q2$, we vary the curvature value. We also modify them to calculate only the results over the last d seconds, at each d seconds. To evaluate the overheads, we measure execution time without monitoring and then with monitoring, but varying the number of queries $|\{Q\}|$ and the interval d, which is considered the same for all queries in $\{Q\}$ in this experiment. The experiments were repeated until the standard deviation of workflow elapsed times was less than 1%. The results are the average of these times within the 1% margin. Figure 18 shows the results, where the blue portion represents the workflow execution time when no monitoring is used and the red portion represents the difference between the workflow execution time with and without monitoring (*i.e.*, the monitoring overhead).

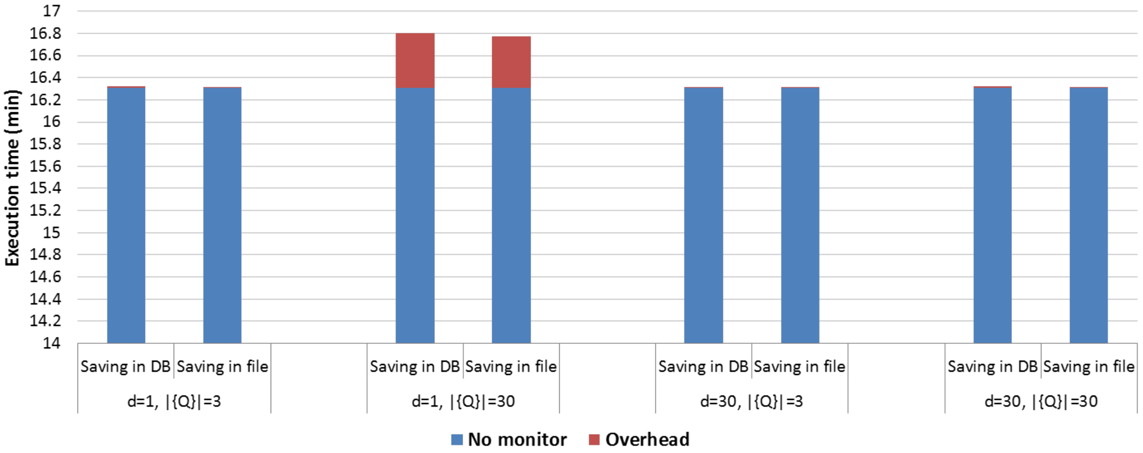


Figure 18. Results of adaptive monitoring overhead.

From these results, we observe that when the interval d is equal to 30s, the overhead is negligible. For 1s interval, the overhead is higher when the number of monitoring threads is also higher. This happens because three queries are executed in each time interval (see Figure 9), for each thread. In the scenarios with 30 threads, there will be 120 queries in a single time interval d. In that case, if d is small (*e.g.*, $d = 1$), there are 120 queries being executed per second, just for the monitoring. The database that is queried by the monitors is also concurrently queried by the SWMS engine, thus adding higher overhead. However, even in this specific scenario that shows higher overhead ($|\{Q\}| = 30$ and $d = 1$), it is only 33 s or 3.19% higher than when no monitoring is used. Most of the real monitoring cases do not need such frequent (every second) updates. If 30s is frequent enough for the user, there might be no added overhead, like in this test case. We also evaluated the same scenarios without storing monitoring results in the wf-Database, but rather appending in CSV files, which is simpler. As Figure 18 shows, the results are nearly the same as in either cases (saving in the wf-Database or saving in CSV files). This suggests storing all monitoring results in the wf-Database at runtime, which enables users to submit powerful queries as the monitoring results are generated, with all other provenance data. This would not be possible with a solution that appends data to CSV.

8. Related Work

Considering our contributions, we discuss the SWMSs with respect to human adaptation (especially data reduction), online provenance support, and monitoring features.

We proposed a data-centric data reduction approach, which requires modifications in the workflow scheduling, since tasks associated to the removed input data should not be executed. Consequently, they should not be a part of the workflow execution plan. To be able to support online human adaptation, the SWMS needs to employ a data-centric execution model. Although online human adaptation is the core of computational steering, there are few SWMSs [29–31] that support it. These solutions have monitoring services and are highly scalable, but do not allow for online data reduction as a mean to reduce execution time. WorkWays [27] is a powerful science gateway that enables users to steer and dynamically reduce data being processed online by dimension reduction or by reducing the range of some parameters, sharing similar motivations to our work. It uses Nimrod/K as its underlying parallel workflow engine, which is an extension of the Kepler workflow system [32]. WorkWays presents several tools for user interaction contributing to human-in-the-loop workflows, such as graphic user interfaces, data visualization, and interoperability among others. However, WorkWays does not provide for provenance representation and users may not define an online query involving simulation data, execution data, metadata, and provenance, all related in a database, which limits the power of online computational steering. For example, it prevents *ad-hoc* data analysis using both domain and workflow execution data, such as those presented in Table 1 and Table 2, which support the user in defining which slice of the dataset should be removed. In contrast, d-Chiron uses an in-memory distributed database system to manage and relate analytical data involved in the workflow execution. Moreover, the lack of provenance data support in WorkWays, either online or *post-mortem*, does not support reproducibility and prevents from registering user adaptations, missing opportunities to determine how specific user interactions influenced workflow results. Another SWMS example is WINGS/Pegasus [33], which focuses on assisting users in automatic data discovery. It helps generating and executing multiple combinations of workflows based on user contraints, selecting appropriate input data, and eliminating workflows that are not viable. However, it differs from our solution in that it tries to explore multiple workflows until finding the most suitable one, whereas we often model our experiments as one single scientific workflow to be fine tuned as the results come out. Also, it does not employ a data-centric execution model and does not aim at providing online computational steering support to actively eliminate subsets of an input dataset, especially based on extensive *ad-hoc* intermediate data analysis online. It has a static execution model, in the sense that the execution is predefined, submitted to the HPC environment, and no online human adaptation is enabled. Additionally, as WorkWays, provenance data is not collected online, nor is it integrated with domain-specific and execution data for enhanced analysis. Likewise, solutions like VisTrails and Kepler do not support data-centric online human adaptations.

Different from those SWMSs, Swift [15] has a data-centric execution model. It has a parallel dataflow programming language that allows to write scripts for distributing tasks across distributed computing resources. It runs multiple programs in parallel as soon as their input data are available. It generates tasks at runtime and, consequently, it has potential to enable online human-adaptation in the execution plan. However, to the best of our knowledge, it does not support user-steered online input data reduction, nor does it store provenance data related to domain data during workflow execution.

Moreover, while our data reduction techniques aim at avoiding data to be generated, there is an intense area of data reduction research focused on reducing data already generated by the simulation. For example, initiatives like CODAR [34] propose data reduction strategies such as dimension reduction, outlier detection and compression also based on online data analyses, which are complementary to our approach.

Although human adaptation is a desired feature that remains an open problem in SWMSs, monitoring is widely supported in several existing SWMSs [4,35]. For example, Pegasus [36] provides a framework to monitor workflow executions and has rich capabilities for online performance monitoring, troubleshooting, and debugging. However, in such solutions, it is not possible to monitor workflow execution data associating to provenance and domain data, or run *ad-hoc* online data queries, as we do using data in the wf-Database. To the best of our knowledge, no related work allows for online data reduction based on a rich analytical support with adaptive monitoring and provenance registration of human adaptations in HPC workflows. These features allow for performance improvements of scientific workflows, while keeping data reduction consistency and provenance queries that can show the history of user-steering actions and results.

9. Conclusion

This work contributes to putting the human in the loop of online scientific workflow executions, especially when users can actively steer and reduce data to improve performance. As a solution to the input data reduction problem, we make use of a data-centric algebraic approach that organizes workflow data to be processed as sets of data elements stored in a wf-Database, managed by an in-memory distributed database system at runtime. We introduced Cut as part of a new class of algebraic operators that only exist because of dynamic human adaptation actions. This is the first work that introduces this representation for dynamic workflow adaptations. Cut is just one among many other user adaptation possibilities that are yet to be explored. We developed a mechanism coupled to d-Chiron, a distributed version of Chiron SWMS, to implement Cut and maintain both data and execution consistency after a reduction, and track provenance of user adaptations. A major challenge to the problem of data reduction is to identify which subset of the data should be removed. To address it, we proposed an adaptive monitoring approach that aids users in analyzing partial result data at runtime. Based on the evaluation of input data elements and its corresponding results, the user may find which subset of the input data is not interesting for a particular execution, hence can be removed. The adaptive monitoring allows users not only to follow the evolution of the workflow, but also to dynamically adjust monitoring aspects during execution. We extended our previous workflow provenance data diagram to be able to represent provenance of the online data reduction actions by users and the monitoring results. Although we implemented our solution in d-Chiron, other SWMS could be used if provenance, execution, and domain dataflow data are managed in a database at runtime.

To validate our solution, we executed a data-intensive parameter sweep workflow based on a real case study from oil and gas industry, running on a 936-cores cluster. A test case demonstrated how the user can monitor the execution, dynamically adapt monitoring settings, and remove uninteresting data to be processed, all during execution. Results for this test case show that the user interactions reduced the execution time by 32% and total amount of data processed by 14%. Although the test case was from the oil and gas domain, other workflow applications could have been used, like the ones discussed in bioinformatics and astronomy domains (Section 4.5). In SciPhy workflow, users frequently run in cloud environments where the longer the workflow takes to execute, the more expensive the final costs will be. Being able to dynamically identify certain combinations of genomic sequences that will make the execution take undesirably longer and remove such combinations online are very beneficial features to SciPhy's users. Whereas in Montage workflow, users can identify certain regions of the outer space that are unlikely to contain celestial objects and remove delimited regions from the input dataset during execution. Thus, as long as users can tell which slice is not interesting, our solution supports dynamic reduction from the input data with no harm to the final results.

To the best of our knowledge, this is the first work that explores user-steered online data reduction in scientific workflows steered by *ad-hoc* queries and adaptive monitoring, while maintaining provenance of user interactions. The results motivate us to extend our solution and explore different aspects that can be adapted based on dataflow online data analysis. Our solution is currently dependent on the users' knowledge to identify correlations between input and output data to determine which subsets are uninteresting. We plan to address *in-situ* data visualization using adaptive monitoring and interactive queries results and develop recommendation models to suggest correlations based on the history stored in the wf-Database to help identifying such correlations. Other future works include: enabling users to set priorities to different slices of the data in a way that the SWMS system will process critic slices before; exploring the potential of the solution in a higher extent; and improving usability of the system by improving the system's interfaces.

Acknowledgments

This work was partially funded by CNPq, FAPERJ and Inria (SciDISC project), EU H2020 Programme and MCTI/RNP-Brazil (HPC4E grant no. 689772), and performed (for P. Valduriez) in the context of the Computational Biology Institute (www.ibc-montpellier.fr). The experiments were carried out using Grid'5000 (https://www.grid5000.fr). The authors would like to thank Andres Codas, Juliana Jansen, and Heloisa Candello from IBM Research for their help on how the system is being used by scientists for data reduction.

References

[1] I. Raicu, I.T. Foster, and Yong Zhao. Many-task computing for grids and supercomputers. *MTAGS*, 1–11, 2008.

[2] J. Dias, G. Guerra, F. Rochinha, A.L.G.A. Coutinho, P. Valduriez, and M. Mattoso. Data-centric iteration in dynamic workflows. *FGCS*, 46(C):114–126, 2015.

[3] E. Deelman, D. Gannon, M. Shields, and I. Taylor. Workflows and e-Science: an overview of workflow system features and capabilities. *FGCS*, 25(5):528–540, 2009.

[4] M. Mattoso, J. Dias, K.A.C.S. Ocaña, E. Ogasawara, F. Costa, F. Horta, V. Silva, and D. de Oliveira. Dynamic steering of HPC scientific workflows: A survey. *FGCS*, 46:100–113, 2015.

[5] S.B. Davidson and J. Freire. Provenance and scientific workflows: challenges and opportunities. *SIGMOD*, 1345–1350, 2008.

[6] F. Costa, V. Silva, D. de Oliveira, K. Ocaña, E. Ogasawara, J. Dias, and M. Mattoso. Capturing and querying workflow runtime provenance with PROV: a practical approach. *EDBT/ICDT Workshops*, 282–289, 2013.

[7] V. Silva, D. de Oliveira, P. Valduriez, and M. Mattoso. Analyzing related raw data files through dataflows. *CCPE*, 28(8):2528–2545, 2015.

[8] R. Souza, V. Silva, Oliveira, Daniel, P. Valduriez, A.A.B. Lima, and M. Mattoso. Parallel execution of workflows driven by a distributed database management system. *Poster in IEEE/ACM Supercomputing*, 1–3, 2015.

[9] R. Souza, V. Silva, A.L.G.A. Coutinho, P. Valduriez, and M. Mattoso. Online input data reduction in scientific workflows. *WORKS*, 44–53, 2016.

[10] K.A.C.S. Ocaña, D. de Oliveira, E. Ogasawara, A.M.R. Dávila, A.A.B. Lima, and M. Mattoso. SciPhy: A Cloud-Based Workflow for Phylogenetic Analysis of Drug Targets in Protozoan Genomes. *Advances in Bioinformatics and Computational Biology*, 66–70, 2011.

[11] J.C. Jacob, D.S. Katz, G.B. Berriman, J.C. Good, A.C. Laity, E. Deelman, C. Kesselman, G. Singh, M.-H. Su, et al. Montage: a grid portal and software toolkit for science-grade astronomical image mosaicking. *International Journal of Computational Science and Engineering (IJCSE)*, 4(2):73–87, 2009.

[12] Det Norse Veritas. Recommended practice: riser fatigue. *DNV-RP-F204*, 2010.

[13] V. Silva, J. Leite, J.J. Camata, D. de Oliveira, A.L.G.A. Coutinho, P. Valduriez, and M. Mattoso. Raw data queries during data-intensive parallel workflow execution. *FGCS*, 75:402–422, 2017.

[14] E. Ogasawara, J. Dias, D. Oliveira, F. Porto, P. Valduriez, and M. Mattoso. An algebraic approach for data-centric scientific workflows. *PVLDB*, 4(12):1328–1339, 2011.

[15] J.M. Wozniak, T.G. Armstrong, M. Wilde, D.S. Katz, E. Lusk, and I.T. Foster. Swift/T: large-scale application composition via distributed-memory dataflow processing. *IEEE/ACM Int. Symp. Cluster, Cloud and Grid Computing*, 95–102, 2013.

[16] M. Zaharia, M. Chowdhury, M.J. Franklin, S. Shenker, and I. Stoica. Spark: cluster computing with working sets. *Proceedings of the 2nd USENIX conference on Hot topics in cloud computing*, 10–17, 2010.

[17] R. Ikeda, A. Das Sarma, and J. Widom. Logical provenance in data-oriented workflows?. *Proceedings of the 2013 IEEE International Conference on Data Engineering (ICDE 2013)*, 877–888, 2013.

[18] D. De Oliveira, V. Silva, and M. Mattoso. How Much Domain Data Should Be in Provenance Databases?. *Proceeding of the 7th USENIX Workshop on the Theory and Practice of Provenance (TaPP 15)*, 2015.

[19] V. Silva, L. Neves, R. Souza, A. Coutinho, D. de Oliveira, and M. Mattoso. Integrating domain-data steering with code-profiling tools to debug data-intensive workflows. *WORKS*, 2016.

[20] J. Dias, E. Ogasawara, D. Oliveira, F. Porto, A.L.G.A. Coutinho, and M. Mattoso. Supporting dynamic parameter sweep in adaptive and user-steered workflow. *WORKS*, 31–36, 2011.

[21] R. F. da Silva, R. Filgueira, I. Pietri, M. Jiang, R. Sakellariou, and E. Deelman. A characterization of workflow management systems for extreme-scale applications. *FGCS*, 75:228–238, 2017.

[22] G. Juve, A. Chervenak, E. Deelman, S. Bharathi, G. Mehta, and K. Vahi. Characterizing and profiling scientific workflows. *FGCS*, 29(3):682–692, 2013.

[23] GitHub. d-Chiron Repository. Available at: github.com/hpcdb/d-Chiron

[24] M.T. Özsu and P. Valduriez. *Principles of distributed database systems*. 3 ed. New York, Springer, 2011.

[25] L. Moreau and P. Missier. PROV-DM: the PROV data model. Available at: http://www.w3.org/TR/prov-dm Accessed: 1 Aug 2016., 2013.

[26] M. Mattoso, K. Ocaña, F. Horta, J. Dias, E. Ogasawara, V. Silva, D. de Oliveira, F. Costa, and I. Araújo. User-steering of HPC workflows: state-of-the-art and future directions. *Proceedings of the 2nd ACM SIGMOD Workshop on Scalable Workflow Execution Engines and Technologies (SWEET)*, 1–6, 2013.

[27] H.A. Nguyen, D. Abramson, T. Kiporous, A. Janke, and G. Galloway. WorkWays: interacting with scientific workflows. *Gateway Computing Environments Workshop*, 21–24, 2014.

[28] M. Abouelhoda, S. Issa, and M. Ghanem. Tavaxy: Integrating Taverna and Galaxy workflows with cloud computing support. *BMC Bioinformatics*, 13:77, 2012.

[29] R. Reuillon, M. Leclaire, and S. Rey-Coyrehourcq. OpenMOLE, a workflow engine specifically tailored for the distributed exploration of simulation models. *FGCS*, 29(8):1981–1990, 2013.

[30] A. Jain, S.P. Ong, W. Chen, B. Medasani, X. Qu, M. Kocher, M. Brafman, G. Petretto, G.-M. Rignanese, et al. FireWorks: a dynamic workflow system designed for high-throughput applications. *CCPE*, 27(17):5037–5059, 2015.

[31] J. Kephart and R. Das. Achieving Self-Management via Utility Functions. *IEEE Internet Computing*, 11(1):40–48, 2007.

[32] D. Abramson, C. Enticott, and I. Altinas. Nimrod/K: Towards massively parallel dynamic grid workflows. *Supercomputing*, 24:1–24:11, 2008.

[33] Y. Gil, V. Ratnakar, J. Kim, P. Gonzalez-Calero, P. Groth, J. Moody, and E. Deelman. Wings: intelligent workflow-based design of computational experiments. *IEEE Intellig. Sys.*, 26(1):62–72, 2011.

[34] I. Foster, M. Ainsworth, B. Allen, J. Bessac, F. Cappello, J.Y. Choi, E. Constantinescu, P.E. Davis, S. Di, et al. Computing just what you need: online data analysis and reduction at extreme scales. *Euro-Par*, 3–19, 2017.

[35] A. Mandal, P. Ruth, I. Baldin, D. Krol, G. Juve, R. Mayani, R. Ferreira Da Silva, E. Deelman, J.S. Meredith, et al. Toward an end-to-end framework for modeling, monitoring and anomaly detection for scientific workflows. *IPDPSW*, 1370–1379, 2016.

[36] E. Deelman, K. Vahi, G. Juve, M. Rynge, S. Callaghan, P.J. Maechling, R. Mayani, W. Chen, R. Ferreira da Silva, et al. Pegasus, a workflow management system for science automation. *FGCS*, 46(C):17–35, 2015.