Online Input Data Reduction in Scientific Workflows

Renan Souza, Vítor Silva, Alvaro L. G. A. Coutinho, Patrick Valduriez, Marta Mattoso

Online Input Data Reduction in Scientific Workflows

Renan Souza[§,°], Vítor Silva[§], Alvaro L G A Coutinho[§],
Patrick Valduriez[¶], Marta Mattoso[§]
[§]COPPE/Federal University of Rio de Janeiro, [°]IBM Research Brazil, [¶]Inria and LIRMM, Montpellier

ABSTRACT

Many scientific workflows are data-intensive and need be iteratively executed for large input sets of data elements. Reducing input data is a powerful way to reduce overall execution time in such workflows. When this is accomplished *online* (*i.e.*, without requiring users to stop execution to reduce the data and resume execution), it can save much time and user interactions can integrate within workflow execution. Then, a major problem is to determine which subset of the input data should be removed. Other related problems include guaranteeing that the workflow system will maintain execution and data consistent after reduction, and keeping track of how users interacted with execution. In this paper, we adopt the approach "human-in-the-loop" for scientific workflows by enabling users to steer the workflow execution and reduce input elements from datasets at runtime. We propose an adaptive monitoring approach that combines workflow provenance monitoring and computational steering to support users in analyzing the evolution of key parameters and determining which subset of the data should be removed. We also extend a provenance data model to keep track of user interactions when users reduce data at runtime. In our experimental validation, we develop a test case from the oil and gas industry, using a 936-cores cluster. The results on our parameter sweep test case show that the user interactions for online data reduction yield a 37% reduction of execution time.

CCS Concepts

• **Massively parallel and high-performance simulations.**

Keywords

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

1. INTRODUCTION

Scientific Workflow Management Systems (SWMS) with parallel capabilities have been designed for executing data-intensive scientific workflows, scientific workflows for short hereafter, in High Performance Computing (HPC) environments. A typical execution may involve thousands of parallel tasks with large input datasets. When the workflow is iterative, it is repeatedly executed for each element of an input dataset. The more the data to be processed, the longer the workflow may take, which may be days depending on the problem and HPC environment [7]. Configuring a scientific workflow with parameters and data to be processed is hard. Typically, the user needs to try several input data or parameter combinations in different workflow executions. These trials make the scientific experiment take even longer. It is well known that optimizing performance of the parallel system is a way to improve overall workflow execution time, but reducing the input data that was initially planned to be processed is another effective approach to reduce workflow execution time [4].

In scientific workflows, the total amount of data is very large, but not necessarily the entire input dataset has relevant data for achieving the goal of the workflow execution. This is particularly the case when a large parameter space needs to be processed in parameter sweep workflows. There may be slices of the parameter space that will not influence relevant results and thus, as with a "branch and bound" optimization strategy, can be bounded. A similar scenario occurs when the workflow involves a large input dataset. When domain-specialist 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 or not for the final result [14]. Then, based on their domain knowledge, users can identify which subset of the data is not interesting and thus should be removed from the execution by the SWMS, thereby reducing execution time.

Data reduction can be accomplished in at least three different forms. First, users can do it before the execution starts. However, in most complex scenarios, the high number of possibilities makes it impossible to know beforehand the uninteresting subsets, without any prior execution. Furthermore, not only the initial dataset can be reduced, but also the data generated by the workflow, since the activities composing scientific workflows continuously produce significant amounts of partial data that are consumed by other activities. A second form of data reduction is to do it online. When the SWMS allows for partial result data analysis, the user may interact with the generated partial data, find which slice of the dataset is not interesting, and reduce the dataset online. We use the term *online* for the interactions where users are able to inspect workflow execution, analyze partial and performance data, and dynamically adapt (*i.e.*, steer) workflow settings while the workflow is running (*i.e.*, at runtime). The third form of data reduction is by stopping execution, reducing the data offline, and then resuming execution with the reduced dataset. Because of the difficulty in defining the exploratory input dataset and the long execution time of such iterative 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 [7].

Online data reduction has obvious advantages but introduces several problems related to computational steering in HPC environments [14]. First, because of the complexity of their scientific question to address and the huge amounts of data, users do not exactly know beforehand which subset of dataset should be kept or removed. 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 subset of the data. Indeed, this is the main related challenge. Second, if they can find which subset to remove and actually try to remove, the SWMS must guarantee that the operation will be done consistently. Otherwise, it can introduce anomalous data, yielding to no control of data elimination, data redundancy, or even execution crash. Third, in a long run, there may be more than one user interaction, each removing more subsets, at different times. If the SWMS does not keep track of user actions, it may negatively impact the results' reproducibility and reliability. Although data reduction is not new in SWMS [4],

to the best of our knowledge, these problems related to online user-steered data reduction while maintaining data provenance have not been addressed by related works.

Our approach is to represent workflow input datasets as database relations. Each input element from the scientific domain dataset is represented as a tuple of the input relations. When the elements of the input dataset are files, we insert paths to these files. The approach is implemented in Chiron, a parallel SWMS that adopts a tuple-oriented algebraic approach [17]. Chiron has been used to manage workflow applications with user steering in domains, such as bioinformatics [2], computational fluid dynamics [7], astronomy [20], etc. Chiron continuously populates a relational database at runtime to store domain-specific data, workflow execution data, and, more importantly, provenance data, all integrated in the same database available for online queries. In this paper, we use the term *workflow Database* (wf-Database) to refer to this database. In addition to the traditional advantages of managing provenance data in scientific workflows (*i.e.*, reproducibility, reliability, and quality of result data) [3], online provenance data management eases interactive domain data analysis [2][20]. Such interactive flexible data analysis through provenance helps finding which subset of a dataset to be removed. Moreover, execution monitoring is another very desirable feature in any data-intensive system, including SWMS, and can also be used to assist users in addressing the subset to be removed. However, Chiron does not control changes in input datasets, including removing a subset. In this work, we extend Chiron's wf-Database to maintain the provenance of the subsets of the dataset that are removed. Furthermore, we take advantage of a distributed in-memory database system coupled to Chiron, in a version called d-Chiron that is significantly more scalable [21], to address consistency issues with respect to data reduction. We make the following contributions:

- A mechanism coupled to d-Chiron for online input data reduction, which allows users to remove subsets of the dataset to be processed at runtime. It guarantees that both execution and data remain consistent after reduction.
- An extension to a provenance data model (which is W3C PROV compliant) to maintain the history of user interactions when users decide to reduce a dataset during workflow execution.
- An adaptive monitoring approach that combines monitoring and computational steering. It helps users to follow 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 steer the monitoring definitions, such as which data should be monitored and how. Although existing solutions enable workflow execution monitoring [13][16][19], there is no approach to enable users to run monitoring queries that integrate execution, provenance, and domain data, and dynamically adapt these queries online, to the best of our knowledge.

Paper organization: Section 2 gives a motivating example. Section 3 gives the background for this work. We present our online data reduction approach in Section 4 and our adaptive monitoring approach in in Section 5. Section 6 gives the experimental validation. Section 7 discusses related work. Section 8 concludes.

2. MOTIVATING EXAMPLE IN 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 it 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 [6].

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 complex simulation solvers) to consume the files, evaluate the impact on the risers in the near future (*e.g.*, risk of a fracture occurrence), and estimate the risers' fatigue life. Figure 1 presents a scientific workflow composed of seven piped specialized programs (represented by workflow activities) with a dataflow in between.

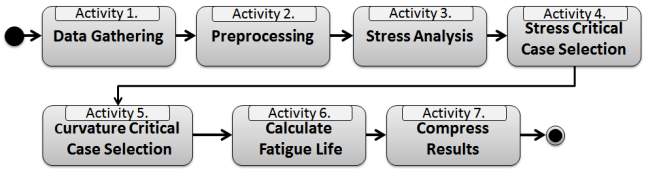

Figure 1. Risers Fatigue Analysis Workflow.

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's, 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 stresses 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 in a good state (no critical stress or curvature values were identified), which are of no interest to the analysis. `Calculate Fatigue Life` (Activity 6) uses previously calculated values to execute a standard methodology [6] 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 intermediary 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 instant in time. 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 > 38m with frequency > 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, more importantly, the overall execution time is reduced. In this paper, we use this workflow in our examples.

3. USER-STEERED WORKFLOWS

Mattoso *et al.* [14] analyze six aspects of computational steering in scientific workflows: interactive analysis, monitoring, human adaptation, notification, interface for interaction, and computing model. Despite their importance, the first three are essential and we mostly focus on those in this work. In fact, human adaptation is definitely 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. For this, interactive analysis and monitoring play an important role to put the human in the loop.

Online provenance data management in SWMS 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.

3.1 Interactive analysis

We address two aspects of workflow data that should be interactively analyzed: *(A) domain dataflow* and *(B) workflow execution information* [14].

A. Domain dataflow. Workflows are composed of activities (scientific programs, scripts, or services) linked as a dataflow. Each activity invocation, or *task*, may consume input datasets and input raw data files and produce output datasets and files. These flows form the domain dataflow. To support domain dataflow interactive analysis, Chiron stores dataflow provenance data in the wf-Database during execution and makes them available for online user queries. Users can then query the wf-Database using a query interface or SQL following PROV-Wf [2], a W3C PROV-compliant data model [15] that specializes PROV entities into domain-data entities to allow for domain dataflow analysis at a finer grain than PROV.

Chiron's tuple-oriented engine first stores input dataset as tuples in the wf-Database. In parameter sweep, tuples are data values from the Cartesian product of input parameters. Then, each task consumes input tuples retrieved from this database, executes them, and then stores the generated output tuples in the wf-Database immediately after task execution, adequately maintaining the data relationships to the input tuples. The workflow activities that generated the tuples are also stored in the wf-Database and linked accordingly. Large raw data files consumed or produced by each task are not stored in the wf-Database, but are rather linked to them, for file flow management.

Thus, Chiron enables online fine-grained domain dataflow analysis [2] as well as the analysis of related domain raw data files through file flow relationships [20]. Table 1 shows some useful queries for the riser fatigue analysis workflow involving domain data and provenance dataflow analysis. The corresponding generated SQL, as well as the relational schema, are in http://github.com/hpcdb/d-chiron. These queries reflect typical user interactions. When these workflows are executed as scripts, without Chiron's support, users look for files in their directories, open files, and try to do this analysis in an *ad-hoc* way, frequently writing programs to "query" these result files. Often they interrupt the execution to fine tune input data and save execution time.

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 meshes files related when computed fatigue life based on stress analysis is lower than 60?

For Queries $Q1$-$Q4$, the SWMS needs to store the history of the tuples generated in Activities 4 and 5 since the beginning of the flow, adequately linking each tuple flow in between. For example, environmental conditions ($Q1$) and 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 tuples from Activity 4 or 5 to tuples from Activity 1, provenance data relationships are required.

B. Workflow execution information. 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 HPC execution. Users may want to interactively investigate how many parallel tasks each node is running. Moreover, tasks can run domain applications that result in errors. If there were thousands of tasks in a large execution, how to determine which tasks resulted in domain application errors and what the errors were? This also eases debugging, an important feature to be provided in large parallel executions. Furthermore, performance data analysis is very useful. Users are frequently interested in knowing how long tasks are taking. All this workflow execution information is important to be analyzed and can deliver much more 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 demand combining different tools and writing specific analysis programs [20].

To support all this, Chiron's wf-Database registers parallel workflow execution data. This means that all necessary execution information for the parallel engine to work are linked to domain dataflow data and managed in the same database. Table 2 shows some provenance queries for the Risers Analysis workflow that link workflow execution data to domain dataflow.

Table 2. Domain data linked to performance data.

$Q5$	Determine the average of each environmental conditions (output of `Data Gathering` – Activity 1) associated to the tasks that are taking execution time more than 2 standard deviations of `Curvature Critical Case Selection` (Activity 5).
$Q6$	Determine the finite element meshes files (output of Preprocessing – Activity 2) associated to the tasks that are finishing with error status.
$Q7$	List the 5 computing nodes with the greatest number of `Preprocessing` activity tasks that are consuming tuples that contain wind speed values greater than 70 Km/h.

3.2 Monitoring

Another important form to help gaining insights from the data is by monitoring in a passive way. It means that users can set up some monitoring analyses and wait for the monitoring results to

be generated. Results might be delivered to end-users as graphical dashboards or three-dimensional *in-situ* scientific data visualizations. As users gain insights from monitoring results, if the SWMS has dynamic analytical support, they can adapt previously set up monitoring configurations or add new monitoring analysis [14]. Also, from these new insights, new data exploration through interactive analysis can be done.

If the SWMS allows for provenance data analysis during workflow execution, monitoring becomes more effective, since the SWMS can exploit the continuously populated wf-Database. By doing this, all the aforementioned data provenance analysis and queries executed by users may be used by a monitoring engine.

3.3 Human adaptation

After users have analyzed partial data and gained insight from them, they may decide to adapt the workflow execution. Adaptation brings powerful abilities to end-users, putting the human in full control of scientific workflow executions. Many aspects can be adapted by humans, but very few systems support human-in-the-loop actions [14]. The human-adaptable aspects range from computing resources involved in the execution (*e.g.*, adding or removing nodes), to checking-point and rolling-back (debugging), loop break conditions, reducing datasets, modification of filter conditions, and very specific parameter fine-tuning.

Populating the wf-Database during workflow execution can help all these aspects. In the Chiron SWMS, it has been shown that it particularly facilitates steering. For example, in [8], it was shown that it is possible to change filter conditions during execution. Also, in [7], it is proposed an algebraic approach to enable steering and dynamic changes of loop conditions during execution of iterative workflows (*e.g.*, modify number of iterations or loop stop conditions), and such approach was evaluated in Chiron. These works show that these adaptations can significantly reduce overall execution time, since domain expert users are able to identify a satisfactory result before the programmed number of iterations. Prior to this work, although [7] has highlighted its advantages, no work has been developed in Chiron to tackle user-steered data reduction online.

Since provenance data is so beneficial, we consider that when a user interacts with the workflow execution, new data (user interaction data) are generated, and thus their provenance must be registered. In a long-running execution, many interactions can occur and many adaptations may be made. If the SWMS does not adequately register the provenance of interaction data, the users can easily lose track of what they have changed in the past. This is critical if the entire computational experiment takes days to finish and many specific adaptations had to occur, since it may be impossible for the users to remember in the last day of execution what they have steered in the first days. Furthermore, adding interaction data to the wf-Database enriches its content and enables future user interaction analysis. One example of how such data can be exploited is that the registered interaction data could be used by artificial intelligence algorithms for understanding interaction patterns and recommend future adaptations. For these reasons, 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.

4. ONLINE DATA REDUCTION

In this section, we show our main contribution. Suppose that after analyzing the monitoring results, a user identifies, within the entire dataset, the subset that contains those values can be removed, hence reducing the dataset.

However, reducing a dataset to be processed has specific constraints that need to be addressed so the execution remains in a consistent state, *i.e.*, with valid data and with guarantee that the execution will not crash. As previously described, Chiron implements a relational data model in a tuple-oriented algebraic approach for scientific workflows [17].

We propose to represent the input dataset as a database *relation*, which is a set of tuples. In the tuple-oriented approach [17], tuples represent a domain-specific dataset to be consumed or produced by a workflow activity. As examples, tuples may be composed of parameter values of a computational model, file paths to a large raw data file (*e.g.*, genomic sequences, finite element meshes, textual data, binary files), or calculated values. In the tuple-oriented approach, removing a subset of the entire dataset to be processed means removing a set of tuples to be consumed by a workflow activity. As a consequence of this removal, the tasks that would process the tuples within the removed set of tuples will not be executed, hence, reducing both workflow execution time and data processing. Data processing reduction becomes more evident if the removed tuples contain paths to large raw data files that would be consumed by tasks if the tuples were not removed. Not only this prevents execution of tasks that would consume uninteresting data, but also the non-executed tasks will not produce more data, reducing overall generated data amount in a workflow execution. Furthermore, if a tuple of a given activity is removed, the following tuples forming the tuple-flow of the next linked activities will not be processed too, reducing data and, more importantly, execution time in cascade.

Addressing which specific subset will be removed is quite important. So, we first formalize the subset to be removed (Section 4.1). In Section 4.2, we describe how we implemented this in d-Chiron, which is a modified version of Chiron that manages the wf-Database in an in-memory distributed database system and is significantly more scalable than the original Chiron [21]. We also highlight that even though we implemented our solutions in d-Chiron, other SWMS could be used. The only requirement is that the SWMS engine needs to manage workflow data as datasets in a tuple-oriented approach and manage domain, provenance, and workflow execution data online in the same database.

4.1 Removing slices of input datasets

In the tuple-oriented approach, to address a subset of the dataset to be removed, we first define a *slice*, which is a subset of tuples to be removed according to a criteria defined by the user. Let R with data schema $\mathfrak{S} = \{C\}$ be the relation that represents a workflow activity input dataset to be reduced. $\{C\}$ is the set of attributes c_j, $1 \leq j \leq |\{C\}|$, from R and each c_j assumes a data type (integer, string, boolean, etc). Moreover, we split R into two subsets P and S, where P is the subset of R containing tuples that have already been processed and S is the subset of R containing tuples that will be processed. Thus, $R \leftarrow P \cup S \mid P \cap S = \emptyset$. P and S have the same schema $\mathfrak{S}$.

Then, we define a slice $\S$ as a subset of S, which is represented as a primary horizontal fragment of S, defined by the *selection* relational algebraic operation σ [18]. Thus, $\S \leftarrow \sigma_F(S)$, where F is the selection formula to obtain the primary horizontal fragment. The formula F may either be a simple predicate (*e.g.*, $c_{10} = 'FATIGUE'$) or a minterm predicate (*e.g.*, $c_1 > 38 \wedge c_2 > 0.1 \wedge c_2 < 1.0$) [18]. Figure 2 shows a workflow example on the left:

Act. 1 consumes input relation R and produces an output relation that also works as an input relation to be consumed by Act. 2, which produces the final output relation. Although in this illustration we show a data reduction in the first activity, we highlight that input data of any workflow activity can be reduced, including intermediary ones, as shown in Section 6, where input data from the second activity is reduced. The input relation R is magnified on the right of Figure 2, where we show the subsets P and S, and the slice § defined by the formula F.

Once the user has selected the slice to be removed (based on user-defined criteria), the slice can be cut off. For this, we define the operation Cut using the *difference* relational algebraic operator, so that $Cut(R, §) \leftarrow R - §$.

By doing so, we ensure that only tuples from S will be removed, since § only contains tuples from S. This is necessary because only tuples that have not been processed yet (*i.e.*, they are "ready" to be processed) can be removed. Otherwise, either the data or the workflow execution may become inconsistent. We note that our solution is applicable to reduce input data of any workflow activity that needs to process a dataset, as long as the SWMS is aware of the data elements composing the dataset.

4.2 Implementation

In this section, we describe how we implement slice removal and cut in d-Chiron. In the SWMS that implements the tuple-oriented approach and manages execution data in the wf-Database, each input tuple (or set of tuples, depending on the dataflow operator) to be consumed is related to a task. For this reason, removing tuples means removing the tasks that would consume the referenced tuples. In the PROV-Wf data model [2], which d-Chiron supports, tasks' data and input tuples are stored in different relations, with a relationship in between. Thus, to implement the set S, as defined previously, we need to semi-join [18] input tuples from the input relation R with tasks in `READY` state in order to only select the tuples that will be processed. Then, to get the identifiers of the ready tasks π_{READY} to be removed, we project over the task identifiers. Using relational algebra:

$$\pi_{READY} \leftarrow \Pi_{task\_id}\left(\sigma_F(R) \ltimes \sigma_{state=READY}(Task)\right),$$

where the formula F is analogous to that in Section 4.1, which is the criteria to select the slice § to be removed. We emphasize that such verifications are necessary to guarantee consistency and are the SWMS's responsibility only, not the users'. The users would only need to specify the formula F to select the slice.

To ease slice removal in d-Chiron, we developed a `Steer` module. With the `Steer` module, users can issue command lines to inform the name of the input relation R and the formula F to select the slice to be removed. Then, the module is responsible for retrieving the identifiers π_{READY} of the ready tasks to be removed (analogous to the set S needed for the slice definition). Instead of physically removing the tasks from the wf-Database, we choose to mark them with the state `REMOVED_BY_USER`. By doing so, we enable these tasks to be later analyzed with provenance queries and to be consistently related within the table `Modified_Task`.

To guarantee consistency, we take advantage of d-Chiron's database system [21]. d-Chiron uses a transaction-optimized in-memory distributed database system that provides atomicity, consistency, isolation, and durability (ACID). In a data reduction, both d-Chiron's engine and the `Steer` module need to concurrently update a shared resource, the `Task` table in the wf-Database. While d-Chiron's engine updates the `Task` table to get tasks for execution and to mark them later as executed, the `Steer` component needs to update the `Task` table to mark the tasks with the identifiers in the π_{READY} slice as removed by user, so that the engine will not get them for execution.

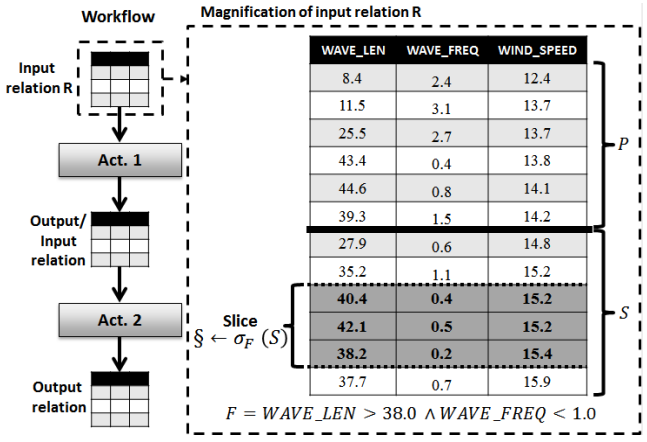

Figure 2. Relation R with subsets P and S, and a slice

The wf-Database tables are distributed, thus making concurrency control of the `Task` table partitions even more complex. In d-Chiron, distributed concurrency control in the `Task` table is outsourced to the distributed database system that guarantees the ACID properties [18]. Thus, concurrency caused by the aforementioned updates is controlled by the database system, guarantying that both execution and data remain consistent.

To store provenance of removed tuples, 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 also keep track of the removed tasks in table `Modified_Task`, which is a table that represents a many-to-many relationship between `User_Query` and `Task` tables. In Section 5.2, we will give the necessary extensions to the PROV-Wf data model implemented in d-Chiron to accommodate `User_Query` and modified tasks.

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. We currently only implemented "Removal" of tuples, but it can be extended in future work.
`user_id`	Relationship with the user who issued the interaction query
`wkfid`	To maintain relationship with the rest of workflow execution data.

5. ADAPTIVE MONITORING

In this section, we combine monitoring and computational steering into an adaptive monitoring approach. Our workflow monitoring approach relies on online queries to the continuously populated wf-Database. Users can set up monitoring queries (as in Table 1 and Table 2) and analyze monitoring results.

In Section 5.1, we present a formal description and describe the implementation in Section 5.2 with the extensions to PROV-Wf to accommodate adaptive monitoring and online data reduction.

5.1 Formal description of adaptive monitoring

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 $d_i > 0$. 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 some interactive provenance data analyses, they can add monitoring queries to $\{Q\}$ in an *ad-hoc* manner. Each d_i can be adapted by users, meaning that users have control of the time frame of each mq_i during execution.

Each mq_i execution generates a monitoring query result set mqr_{it}, $t = \{kd_i | k \in \mathbb{N}_{\geq 0}\}$, at each time interval d_i. We constrain that each mqr_{it} must deliver one column only. If users want more columns, they can write different monitoring queries for each new column. However, the number of rows in the result set is not limited. This means that each monitoring result set mqr_{it} should be either a scalar value or an array.

To improve human-in-the-loop, the end-users have the flexibility to adapt monitoring during workflow execution. To do so, at each 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 amount of time during execution (also configured by the user), the system checks if the user has added new monitoring queries in $\{Q\}$. Our adaptive monitoring approach takes full advantage of the data stored online in the wf-Database. More importantly, it enables users to dynamically steer monitoring settings (including which data will be monitored and how), highly benefiting them in finding uninteresting subsets to be removed.

5.2 Implementation

To implement our approach, we first need to extend the wf-Database schema. To store $\{Q\}$, we add the table `Monitoring_Query`, shown in Table 4.

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 queried
`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.

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. To store monitoring results in the wf-Database, we add another table: `Monitoring_Query_Result`, shown in Table 5.

Table 5. `Monitoring_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]"

Similar to what we did for the Steer module, we also developed the module Monitor to facilitate utilization. The Monitor should start at any cluster node that is able to access the distributed database system and should start after the workflow execution has begun, whenever users want to monitor the workflow execution.

Similar to what we did for the `Steer` module, we also developed the module `Monitor` to facilitate utilization. The `Monitor` should start at any cluster node that is able to access the distributed database system and should start after the workflow execution has begun, whenever users want to monitor the workflow execution.

A command line starts the `Monitor` module that runs in background. It establishes a connection with the distributed database system (connection settings are provided in the XML configuration file). Chiron (and d-Chiron) makes use of this XML 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 to check if a new monitoring query was added. The default value for s is 30s, as the time interval to check if monitoring queries were added or removed. However, users can customize this. 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 the SQL query to 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 `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 3 shows the steps executed at each time interval.

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

Figure 3. Steps executed by each thread within a time interval.

To enable all these monitoring capabilities and human-adaptation, 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.

Another contribution of this paper is that we add three concepts to PROV-Wf [2], which is W3C PROV-compliant [15]. Our main motivations to adhere to the W3C PROV recommendations are to help on query specification, to maintain compatibility between different SWMS and facilitate interoperability between different databases.

These concepts are: `UserQuery`, `MonitoringQuery`, and `MonitoringResult`, as in Figure 4. Using PROV nomenclature, `UserQuery` is a *PROV Activity* that stores the user queries that remove sets of tuples and thus influence the state of the associated tasks (*i.e.*, remove them). `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. EXPERIMENTAL VALIDATION

In this section, we validate our solution (for online data reduction and adaptive monitoring) based on a real data. In Section 6.1, we show the experimental setup, Section 6.2 shows a test case where an expert monitors the execution and removes slices of the dataset. In Section 6.3, we analyze the added overhead.

6.1 Experimental setup

Scientific workflow. As a proof of concept for this work, we use a synthetic parameter sweep workflow of the Riser Fatigue Analysis example (see Figure 1), which is based on a real case study. The workflow manipulates approximately 300 GB of raw data. In all executions, we use the same dataset, which spans over approximately 12,000 data elements to be processed in parallel. Depending on the workflow activity, tasks may take few seconds (*e.g.*, Activity 7) or up to one minute on average (*e.g.*, Activity 3).

Software. In all executions, we use d-Chiron [21], which uses 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 in github.com/hpcdb/d-chiron.

Hardware. The experiments were conducted in Grid5000 using a cluster with 39 nodes, containing 24 cores each (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.

6.2 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 (see from 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 30s. 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 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 5. It presents mqr_{1t} in full black line with square markers and mqr_{2t} in full gray line with triangle markers. These markers determine when the monitoring occurred.

The workflow execution starts at $t = 0$, but only after approximately 150s, the first output results from Activity 6 starts 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 5), 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

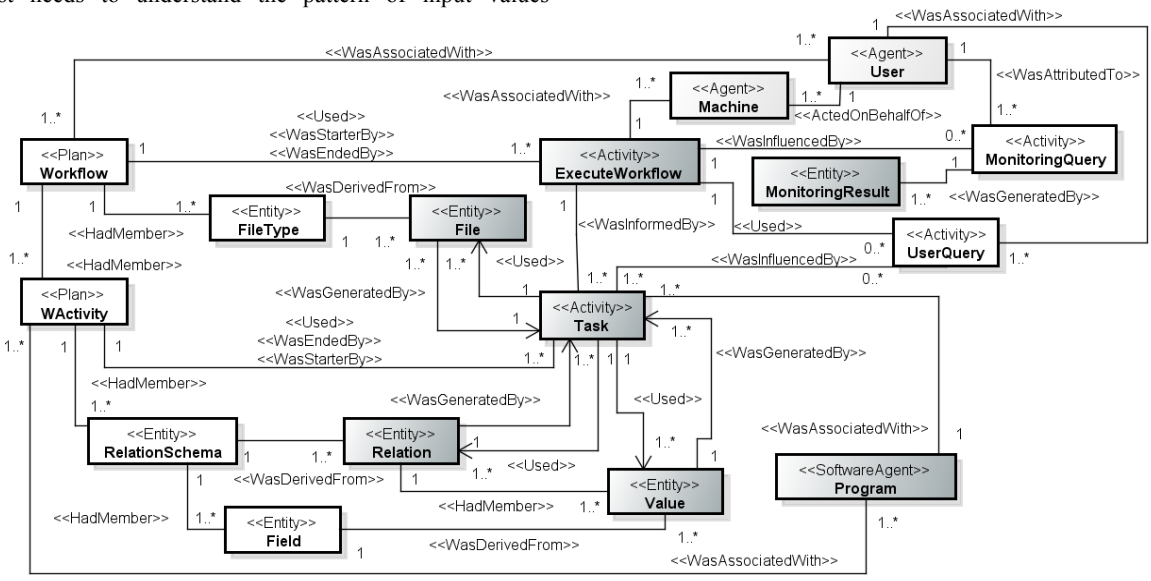

Figure 4. Extended PROV-Wf data module to accommodate modified tasks and monitoring

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 circles in the horizontal axis (Elapsed time) in Figure 5. The exact time a user issued an interaction query is stored in `User_Query` table.

The next markers 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 the monitoring interval time (d_1 and d_2) at runtime, by reducing to 10s 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 10s. We highlight that, although in test use case we are only showing monitoring correlating 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, Peter decides to increase back the monitoring query intervals for both queries to 30s after $t = 270$. He then 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_Task`) its rows with rows in the `Task` table, to consistently keep provenance of which tasks 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 5 shows that some specific interactions imply significant changes in lines' slopes. Queries on the wf-Database can show finer details about how many tuples each user interaction made the SWMS not process, as shown in Table 6. Each issued time follows Figure 5 and is registered with the timestamp of when the first activity started.

Table 6. Provenance of slices removed by the user

Inter act.	Issued time (s)	Slice query	Number of removed data elements
q_1	190	`wind_speed < 16`	623
q_2	310	`wind_speed < 25`	373
q_3	370	`wind_speed < 30`	355
q_4	430	`wind_speed < 34.5`	115
q_5	520	`wind_speed < 35.5`	3

Finally, we run the exact same workflow and input datasets, but with no monitoring or interactions to compare how such slice removals help decrease overall execution time. The workflow with no interaction processes all input data, including those containing wind speed values that lead to risers with low risk of fatigue, which are not considered in Peter's analyses. In total, Peter's steering yields the removal of 1469 input data elements (out of approximately 12,000). This reduces the execution time for this test case by 37% compared with no steering. Furthermore, these removed input data would make the workflow process and generate more raw data files if the input data elements were not removed. By querying the wf-Database in the end of execution, we found that the execution with no user steering processed approximately 300GB of raw data files, whereas with steering the total was 258GB, representing 14% of data reduction.

6.3 Analyzing monitoring overhead

A 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-

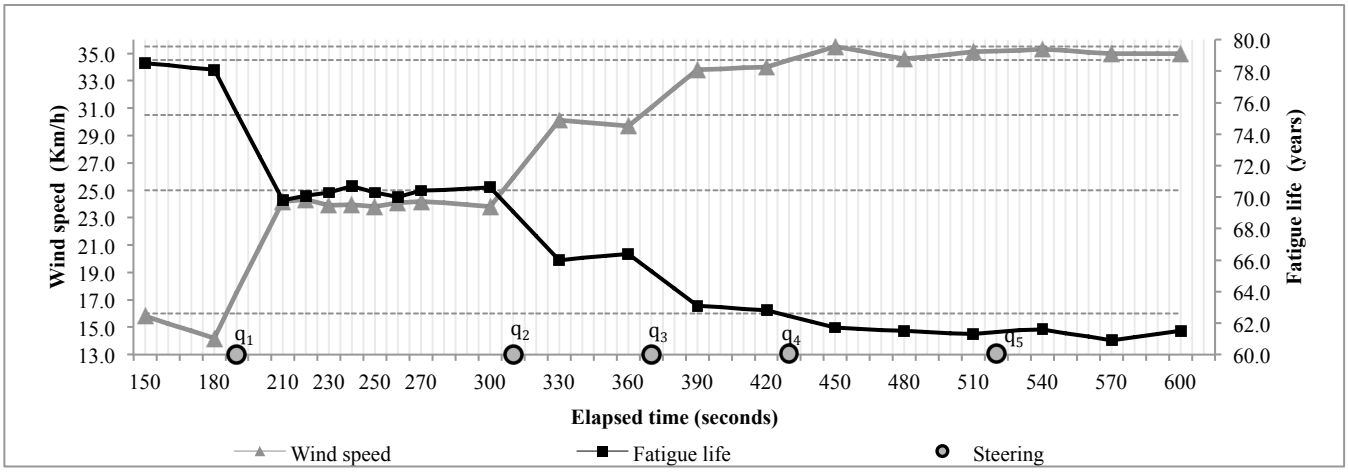

Figure 5. Use case plot to analyze impact of user steering comparing Wind Speed (input) with Fatigue life (output).

Database, which may add overhead. The goal of this experiment is to analyze such overhead.

We set up the `Monitor` module to run queries, which are variations of the queries $Q1$-$Q7$ presented in 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 6 shows the results, where the gray portion represents the workflow execution time when no monitoring is used; and the black portion represents the difference between the workflow execution time with and without monitoring (*i.e.*, the monitoring overhead).

From these results, we observe that when the interval d is equal to 30s, the overhead is negligible. When the interval is 1s, the overhead is higher when the number of monitoring threads is greater. This happens because three queries are executed in each time interval (see Figure 3), 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 frequently 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 33s 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. The results are nearly the same as in Figure 6. This suggests storing all monitoring results in the wf-Database at runtime, which enables users to submit powerful queries as they are generated, with all other provenance data. This would not be possible with a solution that appends data to CSV.

7. RELATED WORK

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

Although online human adaptation is the core of computational steering, there are few parallel SWMS [11][12][19] that support it. These solutions have monitoring services and are highly scalable, but do not allow for online data reduction as a means to reduce overall execution time. WorkWays [16] 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 [1]. WorkWays presents several tools for user interaction in 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 do not have query access to 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, our work uses a robust 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 in detail how specific user interactions influenced workflow results. Another notable SWMS example is WINGS/Pegasus [9], which especially focus 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 the sense 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 processed. Also, it does not aim at putting users in the loop to actively eliminate subsets of an input dataset, especially based on extensive *ad-hoc* intermediary data analysis online. Additionally, as WorkWays, provenance data is not collected online, nor is it integrated with domain-specific and execution data for enhanced analysis.

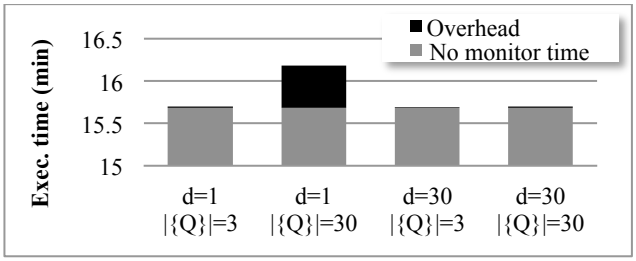

Figure 6. Results of adaptive monitoring overhead.

While human adaptation is less explored in parallel SWMS, monitoring is widely supported in several existing SWMS [13][14]. For example, Pegasus [5] and Triana may be integrated to analytical tools such as Stampede [10][22], which provides a framework to monitor workflow executions and has rich capabilities for online performance monitoring, troubleshooting, and debugging. However, in these solutions, it is not possible to monitor workflow execution data associating them to provenance and domain data, as we do using queries to the wf-Database. To the best of our knowledge, there is no related work that allows for online data reduction based on a rich analytical support with adaptive monitoring and provenance registration of human adaptations in scientific workflows. These features allow for performance improvements of scientific workflows, while keeping data reduction consistency and provenance queries that can show the history of human-in-the-loop actions and results.

8. CONCLUSION

This work contributes to putting the human in the loop of scientific workflow systems, especially when users can actively steer and reduce data online to improve performance. As a solution to the input data reduction problem, we made use of a tuple-oriented algebraic approach that organizes workflow data to be processed as sets of tuples stored in a wf-Database, managed by an in-memory distributed database system at runtime. We developed a mechanism coupled to d-Chiron, a distributed version of Chiron SWMS, which allows for reducing data, while maintaining both data integrity and execution consistency. A major challenge to the problem of data reduction is to address

which subset of the data should be removed. As a solution to this, 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 model 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 the 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 especially remove uninteresting data to be processed, all during execution. Results for this test case show that the user interactions reduced the execution time by 37% comparing with the execution that processed the whole dataset. Although the test case was from the oil and gas domain, any other workflow application could have been used, as long as a domain expert can tell which slice is not interesting, removed 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 by humans based on sophisticated workflow data analysis support. Our solution is currently dependent on the domain expert's knowledge to identify correlations between input and output data to determine which subsets are uninteresting. We plan to address in-situ data visualization based on the adaptive monitoring and interactive queries results and develop recommendation models to suggest correlations based on history stored in the wf-Database. Other future work include: enabling users to set priorities to different slices of the data in a way that the SWMS system will process critic slices before; improving usability of the system by developing intuitive user interfaces to decrease the learning curve, especially related to the query interface, to take full advantage of the wf-Database. We also plan to expand our experiments and analyze how reducing each specific type of data (relation tuples, raw data files not processed and not generated) impact final results.

9. ACKNOWLEDGMENTS

This work was partially funded by CNPq, FAPERJ and Inria (MUSIC 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 the Grid'5000 testbed (https://www.grid5000.fr).

10. REFERENCES

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

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

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

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

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

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

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

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

[9] Gil, Y., Ratnakar, V., Kim, J., Gonzalez-Calero, P., Groth, P., Moody, J., Deelman, E. Wings: Intelligent workflow-based design of computational experiments. *Intelligent Systems*, 26(1):62–72, 2011.

[10] Gunter, D., Deelman, E., Samak, et al. Online workflow management and performance analysis with Stampede. *CNSM*, 152–161, 2011.

[11] Jain, A., Ong, S.P., Chen, W., et al. FireWorks: a dynamic workflow system designed for high-throughput applications. *CCPE*, 27(17):5037–5059, 2015.

[12] Lee, K., Paton, N.W., Sakellariou, R., Fernandes, A.A.A. Utility functions for adaptively executing concurrent workflows. *CCPE*, 23(6):646–666, 2011.

[13] Mandal, A., Ruth, P., Baldin, I., et al. Toward an end-to-end framework for modeling, monitoring and anomaly detection for scientific workflows. *IPDPSW*, 1370–1379, 2016.

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

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

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

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

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

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

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

[21] Souza, R., Silva, V., Oliveira, Daniel, Valduriez, P., Lima, A.A.B., Mattoso, M. Parallel execution of workflows driven by a distributed database management system. *Poster in Supercomputing*, 2015.

[22] Vahi, K., Harvey, I., Samak, T., et al. A case study into using common real-time workflow monitoring infrastructure for scientific workflows. *J. Grid Comput.*, 11(3):381–406, 2013.