Provenance of Dynamic Adaptations in User-steered Dataflows

Renan Souza[1,2], Marta Mattoso[1]
[1]COPPE/Federal University of Rio de Janeiro, Brazil
[2]IBM Research

Abstract. Due to the exploratory nature of scientific experiments, computational scientists need to steer dataflows running on High-Performance Computing (HPC) machines by tuning parameters, modifying input datasets, or adapting dataflow elements at runtime. This happens in several application domains, such as in Oil and Gas where they adjust simulation parameters, or in Machine Learning where they tune models' hyperparameters during the training. This is also known as computational steering or putting the "human-in-the-loop" of HPC simulations. Such adaptations must be tracked and analyzed, especially during long executions. Tracking adaptations with provenance not only improves experiments' reproducibility and reliability, but also helps scientists to understand, online, the consequences of their adaptations. We propose PROV-DfA, a specialization of W3C PROV elements to model computational steering. We provide provenance data representation for online adaptations, associating them with the adapted domain dataflow and with execution data, all in the same provenance database. We explore a case study in the Oil and Gas domain to show how PROV-DfA supports scientists in questions like "who, when, and which dataflow elements were adapted and what happened to the dataflow and execution after the adaptation (*e.g.*, how much execution time or processed data was reduced)", in a real scenario.

Keywords: Computational Steering; Human-in-the-Loop; Dynamic Workflow Provenance.

1 Introduction

It is known that certain actions are better performed by humans than by machines, especially when the actions require very specific domain or application knowledge [1]. Due to the exploratory nature of scientific experiments, this often happens in computational experiments modeled as scientific workflows, where computational scientists (the *users* in this work, who are specialists in application-specific systems, such as engineers, bioinformaticians, data scientists etc.) need to dynamically adapt online workflows while they are running on High-Performance Computing (HPC) machines, *i.e.*, without stopping, modifying, and resubmitting the execution [2].

The data dependencies between programs composing the scientific workflow form the dataflow. Many elements of the dataflow (*e.g.*, data elements, datasets, attribute values, data transformations) can be modified, online, by humans. This occurs in several application domains. For instance, in Oil and Gas HPC simulations where users need to fine tune parameters of a solver[3]; in Machine Learning model training, where data scientists use their knowledge on the data and on the methods to determine better ranges of values for hyperparameters, after analyzing their impact on the performance

(*e.g.*, accuracy); or in Uncertainty Quantification iterative simulations where users control loop stop conditions [4]. Online data analysis and online adaptation steered by humans comprise "computational steering", often referred to as "human-in-the-loop" of HPC applications [2]. In that context, each adaptation occurred for a reason (best known by the user), in a certain time, influenced elements of the dataflow, and had effects in the running workflow, like data or execution time reduction [5]. Therefore, adaptations generate major improvement on performance, resource consumption, and quality of results [6], and thence need to be tracked.

Not tracking such adaptations has impactful disadvantages. It may compromise experiment reproducibility as users hardly remember what and how dataflow elements were modified (especially modifications in early stages), and what happened to the execution because of a specific adaptation. This is more critical when users adapt several times in long experiments, which may last for weeks. In addition to losing track of changes, one misses opportunities to learn from the *user steering data* (*i.e.*, data generated when humans adapt a certain dataflow element) with the associated dataflow. For example, by registering user steering data, one may query the data and discover that when input parameters are changed to certain range of values, the output result improves by a defined amount. Moreover, opportunities to use the data for AI-based recommendations on what to adapt next, based on a database of adaptations, are lost.

Although data provenance in HPC workflows has improved significantly over the past years, adding online data analyses integrating domain and execution data [7] to reproducibility [8], provenance of computational steering in HPC workflows remains an open challenge [6]. Provenance data management and computational steering in HPC are still worlds apart, despite the increasingly need for joint contribution. Indeed, in two recent surveys [6, 9], the authors highlight online provenance capture and human-in-the-loop of HPC simulations as research and development needed. We believe that a provenance representation able to model dynamic interactions in a computational steering system will facilitate data representation, understanding, and standardization among systems. To the best of our knowledge, such model does not exist yet.

In this work, we propose PROV-DfA, a data provenance representation for modeling online human adaptations in HPC workflows, built on W3C PROV standards. It allows for explicit representation of the dataflow and provenance of user-steered dataflow adaptations. PROV-DfA can be implemented in provenance databases of Parallel Scientific Workflow Management Systems (WMS) [2], or computational steering frameworks [10], or standalone HPC applications that allow for user-steered online adaptation. It can represent typical adaptations in HPC applications, while integrating with data for provenance, execution, and domain dataflow, all in a same data representation. We specialize PROV-DfA for provenance parameter tuning, loop control of iterative simulations, and data reduction. To validate our approach, we explore a case study in an Oil and Gas HPC workflow, where the user adapted online elements of the dataflow. We show how those adaptations can be represented using PROV-DfA to answer "who", "what", "when", and "how" queries in a relational provenance database to show, for example, the impact on the results after specific dataflow adaptations.

Paper organization. Related work is presented in Section 2 and background in Section 3. PROV-DfA is presented in Section 4 and in Section 5 we specialize it for provenance of three dataflow adaptations. Section 6 shows the case study. Section 7 concludes.

2 Related Work

As mentioned in introduction, recent surveys [6, 9] bring up challenges of runtime provenance and human-in-the-loop of HPC workflows. Also, Atkinson *et al.* [11] discuss the future of scientific workflows, and they mention that "*monitoring and logging will be enhanced with more interactive components for intermediate stages of active workflows."* As a result, we found no related work for provenance representation of human-in-the-loop of HPC workflows. Thus, we analyze computational steering works that could highly benefit from provenance representation of human adaptation in dataflows.

Long lasting scientific applications require user steering [2, 10]. BSIT [12] is a platform tailored for seismic applications that supports adaptations in parameters, programs, datasets. Few parallel WMSs support human adaptation [13–15], but no provenance of adaptation. Chiron WMS [4, 5] enables users to change filter values, adapt loop conditions of iterative workflows, and reduce input datasets. These works show that online adaptations significantly reduce overall execution time, since users can identify a satisfactory result before the programmed number of iterations.

WorkWays [16] is a science gateway that enables users to dynamically adapt the workflow by reducing the range of parameters. It uses Nimrod/K as its underlying parallel workflow engine, which is an extension of the Kepler workflow system [17]. It presents tools for interaction, such as graphic interfaces, data visualization, and interoperability among others. WINGS [18] is a WMS concerned with workflow composition and its semantics. It focuses on assisting users in automatic data discovery. It helps to generate and to execute multiple combinations of workflows based on user constraints, selecting appropriate input data, and eliminating workflows that are not viable.

Stamatogiannakis *et al.* [19] propose a provenance-based representation and analysis for unstructured processes, including representation of user interactions. However, their target applications are unstructured processes like editing in a content management system, which differ from our target HPC workflows applications, which are considered structured processes.

Bourhis *et al.* [20] propose a provenance-based solution for data-centric applications supporting queries like "*why such result was generated?*", "*what would be the result if an application logic is modified?*", and "*how can a user interact with the application to achieve a goal?*", in the context of users interacting with the application in a "what-if" manner. However, no online user-steered dataflow adaptation in HPC workflows is tackled. Finally, we envision that AI-based systems recommending on what to adapt next [21], could highly benefit from a provenance database containing human user steering data to improve their models.

3 Workflows, Computational Steering and Data Provenance

3.1 Dataflow-oriented Approach and Runtime Provenance

HPC computational experiments are often modeled as scientific workflows. While workflows are related to the execution flow control between chained activities (*e.g.*, scientific programs, processes, scripts, functions or parts of programs) [9], in dataflows datasets are transformed by the chaining of data transformations [22]. A workflow W has an associated *dataflow* D, which has a composition of n *data transformations* (DT),

so that $D = \{DT_1, \ldots, DT_n\}$. Each DT_y, $1 \leq y \leq n$, is executed by a workflow activity, and consumes or produces *datasets*. Datasets are further specialized into *Input Datasets* (I_{DS}) and *Output Datasets* (O_{DS}). Each DT_y consumes one or more I_{DS} and produces one or more O_{DS}. Let $I_y = I_{DS1} \cup \ldots \cup I_{DSu}$ be a set containing all I_{DS} consumed by the DT_y and $O_y = O_{DS} \cup \ldots \cup I_{DSv}$ be a set containing all O_{DS} produced by the DT_y. Then, we have adapted from [22, 23]:

$$O_y \leftarrow DT_y(I_y), \text{for all } DT_y \text{ of the dataflow } D.$$

Moreover, datasets are composed of data elements. Data elements in a given dataset DS have a data schema $\Sigma(DS) = \{attribute_1, \ldots, attribute_u\}$. The schema can be further specified as: $\Sigma(I_{DS}) = \{F_I, V_I, P_I, L_I\}$ and $\Sigma(O_{DS}) = \{F_o, V_o, C_o, L_O\}$, where:

- F_I and F_O contain attributes that represent pointers to input and output files, respectively. These files are often large raw (textual, imagery, matrices, binary data, etc.) scientific datasets in a wide variety of formats depending on the scientific domain (*e.g.*, FITS for astronomy, SEG-Y for seismic, NetCDF for fluid simulations).
- V_I and V_O contain attributes for extracted data or metadata from input and output files, respectively. In case of output data, some applications write calculated values, like the main results of a data transformation into files and they often need to be tracked. V_O represents these special resulting extracted data, which are often scalars, useful for domain data analyses [7, 22, 24]. V_I and V_O can be seen as a view over the actual large raw datasets, as users can have a big picture of the content of the large datasets through them.
- P_I contains attributes for general purpose input parameter values of the data transformation. For example, numerical solver parameters, thresholds, and any other parameter that can be adjusted.
- L_I contain attributes used in the data transformation in case it evaluates a loop [4]. Several applications modeled as scientific workflows have an iterative workflow execution model. Examples include uncertainty quantification and solvers from the Oil and Gas industry [4, 9]. In such workflows, typically there are loops like "*while e > threshold"* or "*while i < max"*. While P_I are for general purpose parameters, L_I contains parameters that are used for loop-stop conditions (*e.g.*, "*max"*, *"threshold"*).
- $L_O$contain output values related to an iteration in case of data transformations that evaluate a loop. In that case, each iteration may be modeled as a loop evaluation execution and produces an attribute value that has the current iteration counter.
- C_O contain attributes for any output values that are explicit data transformation results. For example, besides large scientific data files produced by data transformations, they may produce output quantities, often scalar values or simple arrays that are very meaningful for the result. Since they may be of high interest for the user, these values are typical provenance data that need to be registered.

A schema of a dataset *DS* may not have all these attributes, *i.e.*, they are optional. For example, if a data transformation consuming a dataset *DS* does not evaluate a loop, $\Sigma(DS)$ does not contain L_I or L_O.

Several real HPC workflows have been modeled and specified as previously described, allowing for enhanced provenance data representation [5, 7, 22, 24]. Thus, in addition to well-known advantages of collecting provenance in HPC workflows, such as for experiments' reproducibility and results' reliability [8], runtime provenance aug-

ments online data analytical potential and is especially useful for long-running workflows [2, 4, 5]. In addition to data analyses via *ad-hoc* analytical queries, visualization tools (*e.g.*, ParaView Catalyst) may be coupled to applications querying the database for a graphic view of the execution [24]. Based on online data analyses, the user may dynamically adapt dataflow elements, such as parameters, input data etc. [5]. This is known as computational steering or "human-in-the-loop" of HPC applications.

3.2 A Diagram for Runtime Provenance in HPC Workflows

In a previous work [25], we presented PROV-Wf, which is a PROV-DM [26] specialization. PROV-Wf models workflow provenance, domain-specific, and execution data, all in a same representation. ProvONE [27] has been compared to PROV-Wf in a previous work [28]. It has been implemented in provenance databases of existing WMSs, in real-world workflows [4, 5, 22].

More recently, we extended PROV-Wf into PROV-Df to explicitly represent the dataflow of a workflow [22]. Even without a WMS, runtime provenance can be extracted and integrated to domain data by instrumenting an application. Collecting provenance in standalone HPC applications without a WMS is desired, as there are applications that already employ highly efficient parallel libraries and the WMS scheduling conflicts with the HPC application execution control [24]. A diagram of PROV-Df extended, in this work, for registering human actions, is presented next (Section 4).

In this paper, we use "`prov:`" namespace to indicate PROV classes or relationships. Each `ExecuteDataTransformation` consumes (`prov:used`) and produces (`prov:wasGeneratedBy`) `AttributeValues`. These values may have been extracted by an `ExecuteExtractor` [7]. Data elements compose the dataset (`Dataset`). For prospective provenance, the dataset has an associated `DatasetSchema`, which is composed of `Attribute`. Attributes describe the `AttributeValues` generated during execution. They have a data type (integer, text etc.) and may have extra fields in the `Attribute` class to allow for attribute specification (*i.e.*, determine if the attribute is in $\{F_I, F_O, V_I, V_O, P_I, L_I, L_O, C_O\}$). Such specifications enrich domain data analyses and allow for identifying attributes that can be adapted. Data about execution, such as duration and performance data (CPU, memory), linkage to subsequent and previous executions, and their related prospective provenance can be stored relating to instances of `ExecuteDataTransformation`.

We put this provenance representation into practice in a Bioinformatics HPC workflow to answer "what", "when", and "how" questions , useful for the bioinformatician [7]. She could query output domain data extracted from produced raw datasets, and relate domain data to performance data. However, despite the effort for data provenance in HPC workflows, there is no provenance representation for user steering data.

4 Provenance of Dynamic Adaptation in User-steered Dataflows

During the execution of an HPC workflow, users analyze elements of a dataflow to steer the execution. In this work, we introduce PROV-DfA by specializing provenance data model classes to represent these dynamic adaptations. Instead of creating a completely new provenance model, we first begin by consolidating a base model using several past contributions to PROV-Wf and PROV-Df [5, 7, 22, 25] to build into PROV-

DfA). PROV-DfA adds provenance of online dataflow adaptations to these previous PROV-extended models.

PROV-DfA introduces the classes `SteeringAction`, `Analysis`, `Adaptation`, and `Adapter`; and the relationship `WasSteeredBy`. We use a UML class diagram, where the `<<stereotypes>>` in classes specify PROV super-classes (mainly `Agents`, `Entities`, and `Activities`) and between classes specify relationships. Classes in white background represent prospective provenance, whereas in gray represent retrospective provenance. `prov:Entity` in yellow means that classes in PROV-DfA that are subclasses of `prov:Entity` (prospective or retrospective) can be used in place, as we explain next.

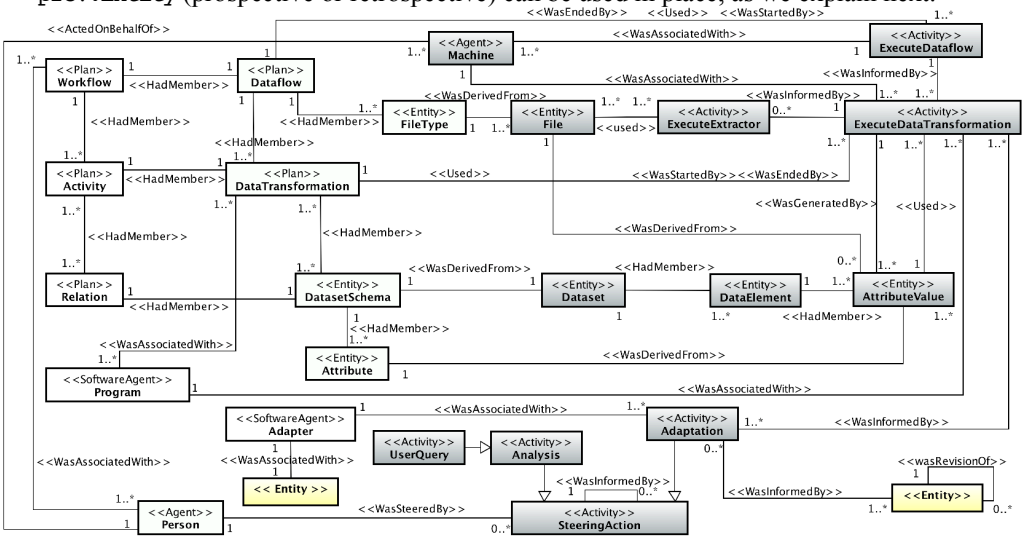

Figure 1. PROV-DfA overview. A larger visualization is on GitHub [29].

Adaptation is a steering action performed by a human that causes a change in the flowing data elements in the dataflow. In PROV-DfA, it is represented by `Adaptation`, a subclass of `SteeringAction`, subclass of `prov:Activity`. An adaptation was steered by (`wasSteeredBy`) a `prov:Person`, occurred at a specified time (`prov:startedAtTime`), had an adaptation characteristic (`adaptationCharacteristic`) that can be "update" or "insert/delete". Users may add a plain description to the adaptation, to describe what was going on in the experiment when they decided to perform a specific change. Also, as inherited by `SteeringAction`, an `Adaptation` may have been informed by a previous `Adaptation`, hence the auto-relationship `prov:wasInformedBy`. This is the case, for example, of a rollback adaptation, requested by a user, that happened right after the user modified parameters in a simulation, which is another adaptation.

Since adaptations in the dataflow occur while the workflow is executing, it is important to keep track of the execution state. The most representative PROV-DfA activity that represents the execution state is `ExecuteDataTransformation`. When an adaption occurs, these instances carry information about time, pointers to domain data values being consumed or produced, computational resources being consumed, etc. Thus, being able to track which specific data transformation was running at the moment of the adaptation may be very useful for extended analyses that integrates adaptation with

provenance, domain, and execution data. For this, we relate which ExecuteDataTransformation instances were influenced (prov:wasInformedBy) by adaptations. How adaptations relate to ExecuteDataTransformation, as well as how prov:Entities are affected depend on characteristic of the online adaptation, as explained next.

Adapter is a software component that knows how to adapt the elements of the dataflow in a running workflow, making it a subclass of prov:SoftwareAgent. In any case, PROV-DfA is just responsible for registering the actions of an Adapter software. Thus, when the user decides to adapt an element of the dataflow, the Adapter is responsible for modifying the requested element. Any information that describes the Adapter software (*e.g.*, which element of the dataflow it adapts, where the program can be located, how it can be invoked etc.) may be stored relating to the Adapter class. Adapter relates to classes that are subclasses of prov:Entity and to the adaptation itself (via prov:wasAssociatedWith).

Characteristics of Online Adaptation. Adaptations may have a characteristic of either update (we say *U-adaptation*) or insert/delete (*I/D-adaptation*).

- *U-adaptations* are updates where the user adjusts, tunes, or modifies one or more dataflow elements. Examples are parameter tuning, loop control adaptations, etc. In PROV-DfA, when the user performs a U-adaptation, a new instance of Adaptation is created. Also, a new instance of one of the prov:Entity subclasses in PROV-Df is created (*e.g.*, AttributeValue, DataTransformation etc.) containing the new data, which will replace the old data in the dataflow. The newly created entity is related (prov:wasInformedBy) to the adaptation. Moreover, the newly created data is related to the old one via prov:wasRevisionOf, so that the track between the new and old data is maintained. Additionally, to relate the adaptation with execution state, PROV-DfA relates (prov:wasInformedBy) the ExecuteDataTransformation instances that were in "*running*" state at the moment of the adaptation. Finally, Adapter is related to the prospective entity (*e.g.*, Attribute, DataTransformation) that specifies the entity adapted.
- *I/D-adaptation*s are steering actions that cause addition or deletion of data elements in the dataflow. Examples are data reduction or extension, data transformation or attribute addition or deletion etc. A new instance of Adaptation is created and there is a relationship (prov:wasInformedBy) between the Adaptation and the added or deleted instances of a prov:Entity subclass. In case of deletions, the entity is not physically deleted from the provenance database, for the sake of provenance. Rather, it is assumed that when an Adaptation is a deletion, the deleted instance is logically deleted from the dataflow. This enables tracking entities deleted online. Since adding or deleting elements affects the execution, the instances of ExecuteDataTransformation directly affected to the added or deleted elements of the dataflow are related (prov:wasInformedBy) to the Adaptation instance. For example, in a data reduction [5], data transformations that were supposed to execute were not executed because of a dynamic adaptation. These instances of DataTransformationExecution not executed are related to the adaptation. Finally, Adapter and Adaptation are related like in U-adaptations.

Furthermore, to use PROV-DfA in a real use case, it is expected that the user will work in collaboration with a data specialist, especially in PROV concepts. Together they specialize the diagram for the domain and application in use, and add provenance capture calls to the simulation via code instrumentation. Users analyze the data via provenance queries together with domain, execution, and user steering data.

In summary, in PROV-DfA, an `Adaptation` is a `prov:Activity` steered by a `prov:Person`, which influenced instances of classes that are subclasses of `prov:Entity`, and influenced instances of `ExecuteDataTransformation`. The `Adapter` program relates to the prospective entity being adapted and to the adaptation.

5 Specializing PROV-DfA Concepts

In this section, we specialize PROV-DfA concepts to represent online parameter tuning, changes in loop control, and data reduction as PROV-DfA's U and I/D-adaptations. We assume that there is a computational steering framework, such as the ones surveyed by Bauer *et al.* [10], or an underlying WMS engine, such as the ones surveyed by Mattoso *et al.* [2], or a standalone program adaptable online, as we show in a previous work [3].

5.1 Simulation Parameter Tuning

Parameter tuning refers to the action of steering parameters of a data transformation in a dataflow, like numerical solver parameters or machine learning model hyperparameters. In PROV-DfA, `ParameterTuning` is a specialization of `Adaptation`. Parameter tunings are adaptations in attribute values (`AttributeValue`) that are related to data elements (`DataElement`) related to I_{DS} (`Dataset`) of a certain data transformation (`DataTransformation`). The attribute value modified must have been derived from (`prov:wasDerivedFrom`) an `Attribute` whose attribute specification is P_I.

It is a U-adaptation. As such, a new instance of `ParameterTuning` is created and related to the new instance of its adapted entity, *i.e.*, `AttributeValue`, with the new value for the parameter. The new value is related to the old one via `prov:wasRevisionOf`. `ExecuteDataTransformation` instances running at the moment of the adaptation are related to the `Adaptation` instance. Finally, since users tune parameters of data transformations, the `Adapter` relates to the `DataTransformation` associated to `DatasetSchema` that had the `Attribute` modified.

5.2 Online Adaptation of Iterative Simulations

Workflows with an iterative workflow execution model have data transformations that evaluate loops. Using the dataflow-oriented approach concepts (Section 3.1), values for these loop-stop conditions may be modeled as an attribute in L_I of a data transformation that evaluates a loop and the iteration counter can be modeled as an attribute in L_O of the data transformation. Moreover, each iteration generates an instance in `ExecuteDataTransformation` for the loop evaluation. During execution of each iteration, a relationship between the output of this data transformation, containing the current iteration value, and the `ExecuteDataTransformation` instance is particularly useful for such workflows, as it identifies a specific part of the workflow execution, and often users can analyze results as the workflow iterates. Such control information is important for the adaptation, as users can associate their specific actions with execution data, such as which point in workflow elapsed time that action happened or what memory/CPU consumption were. In complex iterative simulation, capturing data at each iteration may be managed *in transit* by an efficient database management solution. In a recent work [24], we show an efficient database implementation using an analytics-optimized DBMS and asynchronous provenance capture, including extractions from large domain

raw data files (like metadata V_I and V_O), and related to provenance data in a real iterative HPC simulation. The overall overheads accounted for less than 1% of simulation time and added data, which is considered negligible.

Therefore, in PROV-DfA, it is represented as `LoopAdaptation`, a subclass of `Adaptation`. Similarly to `ParameterTuning`, its instance is related to the new instance of `AtrributeValue`, containing the new value for the loop control condition, relating (`prov:wasRevisionOf`) to the old one. The adapted instance of `AttributeValue` must be derived from an `Attribute` whose attribute specification is L_I. Additionally, the generated `ExecuteDataTransformation` instance related to the output of the last iteration (*i.e.*, last execution of the data transformation for loop evaluation) is related to the `LoopAdaptation` instance. Finally, the adapter must be able to dynamically modify the data transformation that represents the loop evaluation. That is, `Adapter` in this case relates to `DataTransformation`.

5.3 Data Reduction

Online user-steered data reduction are very useful for reducing execution time and amount of data to be processed during a simulation [5]. `DataReduction` is a subclass of `Adaptation`. In the dataflow-oriented approach, the datasets stored as large raw data files to be processed by a data transformation are represented by attributes composing data elements in an I_{DS}. Data files are represented as pointers in F_I, whereas V_I contain extracted domain values from those files specified in F_I. An approach to reduce data is to specify a criteria based on V_I values to eliminate files in F_I to be processed, enabling the adapter program to logically delete data elements in the I_{DS}. This makes the HPC application not to execute the data transformations for the removed elements [5].

Analogously, in PROV-DfA, reducing data means logically removing instances of `DataElement` (and consequently `AttributeValues`) of a `Dataset` (I_{DS}). This can be the result of an I/D adaptation. Thus, there is a relationship (`prov:wasInformedBy`) between the removed instances of `DataElement` and `AttributeValue` and the adaptation. The `ExecuteDataTransformation` instances that would use (`prov:used`) the removed `AttributeValue` instances are related to the `DataReduction` instance. Additionally, the criteria to remove data elements [5] is stored within the adaptation instance. Finally, as users remove data elements in I_{DS}, the adapter is related to the `DataTransformation` associated to the `Dataset` that had the `DataElement and AttributeValues` removed.

6 Case Study

In this section, we present PROV-DfA being used in a real case study in the Oil and Gas domain. In a previous work [24], we applied the domain dataflow-oriented approach (Section 3.1) in an HPC turbidity currents simulation, modeled as an iterative scientific workflow. Parts of the simulation code were identified as workflow activities, modeled as data transformations chained in a dataflow. Data and metadata extractors were developed, and the simulation source code was instrumented to call these domain values extractors, together with provenance data collectors, to populate the datasets in a provenance database at runtime. In Figure 2, we show large raw input files (with mesh data) stored on disk, with pointers in the solver I_{DS}. The solver I_{DS} has over 70 parameters (*i.e.*, P_I attributes), among which only 2 are displayed in the figure (flow linear

and non-linear tolerance). All these solver parameters are extracted from a configurations file, which is read at each iteration. Yet, the maximum number of iterations (`t_max`) is a L_I attribute of the data transformation solver. Some metadata (V_I) are extracted from input raw files at runtime to facilitate tracking their contents while they are processed. Elements of O_{DS} of each data transformation are also collected (via raw data extractors and source code instrumentation) and stored in the database. For example, the solver O_{DS} contains calculated values, such as linear and non-linear results, as well as the current time iteration value. Moreover, the simulation was coupled to data analysis tools for *in-situ* data analysis while the workflow runs [24]. The entire simulation using 3D real data lasts for weeks, making online data analysis a requirement.

In addition to online data analyses, adapters were developed to enable online adaptation of the dataflow. Even though the user could adapt the running dataflow, the adaptations were not being tracked. There were several adaptations during the simulation, and the user lost their track, jeopardizing the experiment's reproducibility and results reliability, and missing opportunities to learn from the adaptations.

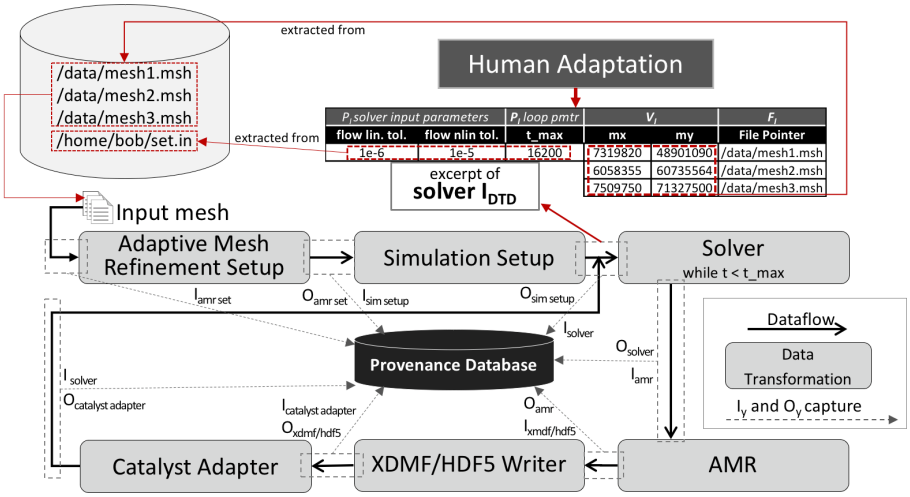

Figure 2. Dataflow in the turbidity currents simulation [24].

We developed a first prototype to instrument the source code of the simulation adapters to collect provenance of adaptation and store in a relational database [3]. However, we developed an *ad-hoc* provenance data model to represent a specific type of adaptation, *i.e.*, tuning some simulation parameters. In this section, we explore this case study to show parameter tuning and data reduction using PROV-DfA.

In Figure 3, we present a visualization of an excerpt of the data in a provenance database implementing PROV-DfA. It shows a user tuning the *flow linear tolerance* parameter from `1e–5` to `1e–3` and a data reduction with criteria "`mx < 7e6`".

Using data in a relational provenance database implementing PROV-DfA, users can run the following queries (their SQL codes are on GitHub [29]).

Inspecting parameter tunings ("*who*", "*when*", "*what*"). *How many tunings did I do? Which parameters did I change? What were the values when I changed and what values did I change into? When did each adaptation happen?*

Understanding consequences of a tuning ("how"). *In parameter tuning 3, how was the main solver output values 10 iterations before and after?*

Data reduction ("*how*", "*which*"). *On average, how long iterations were lasting before and after I reduced input files from the input data? Which files were affected?*

These queries show the potential of PROV-DfA for provenance databases keeping track of online dataflow adaptations in computational steering HPC workflows.

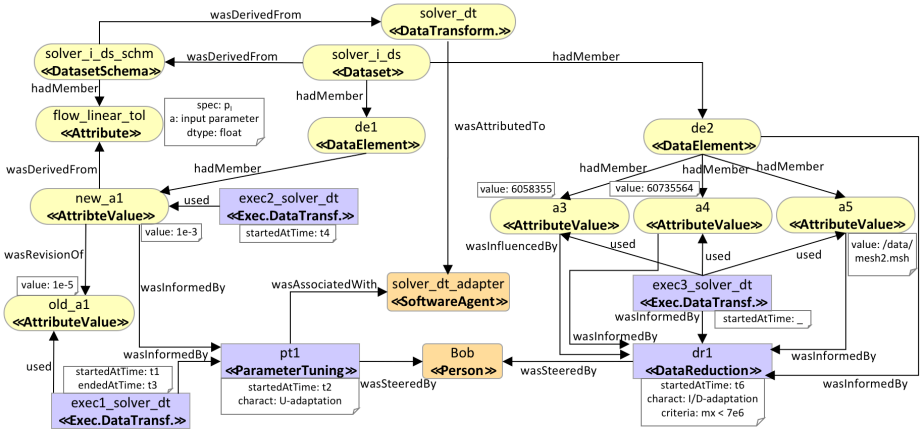

Figure 3. Visualization of data using PROV-DfA.

7 Conclusion

In this work, we presented PROV-DfA, an extension of W3C PROV for provenance of dynamic adaptations in user-steered dataflows. Recent surveys [6, 9] call for research and development in human-in-the-loop of HPC workflows and dynamic data provenance. We believe PROV-DfA is an important step towards modeling provenance of dynamic adaptations in computational steering. To the best of our knowledge, no such model exists yet. Different dynamic dataflow adaptations may be modeled as PROV DfA's U- or I/D-adaptations. We showed it being used for modeling the track of parameter tuning, loop control of iterative simulations, and data reduction steered by users. We queried a provenance database implementing it to answer "who", "what", "when", "how" queries. In the context of computational steering and provenance, our approach contributes for reproducibility, results' reliability, online results understanding as consequences of adaptations, and adds a potential for users or AI-based systems to learn from dynamic interaction data. For future work, we plan to explore PROV-DfA to model the track of other dynamic adaptations and extend it with online data analyses steered by users. We plan to integrate it to ProvONE [27] as well. We expect it can be adopted by WMSs, computational steering frameworks, or standalone HPC applications with steering capabilities that need to keep track of human interactions.

Acknowledgement

This work was partially funded by CNPq, FAPERJ and HPC4E (EU H2020 and MCTI/RNP-Brazil).

References

1. Jagadish, H.V., Gehrke, J., Labrinidis, A., Papakonstantinou, Y., Patel, J.M., Ramakrishnan, R., Shahabi, C.: Big data and its technical challenges. Comm ACM. 57, 86–94 (2014).
2. 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).
3. Souza, R., Silva, V., Camata, J., Coutinho, A., Valduriez, P., Mattoso, M.: Tracking of online parameter tuning in scientific workflows. In: WORKS in ACM/IEEE Supercomputing Workshops (2017).
4. Dias, J., Guerra, G., Rochinha, F., Coutinho, A.L.G.A., Valduriez, P., Mattoso, M.: Data-centric iteration in dynamic workflows. FGCS. 46, 114–126 (2015).
5. Souza, R., Silva, V., Coutinho, A.L.G.A., Valduriez, P., Mattoso, M.: Data reduction in scientific workflows using provenance monitoring and user steering. FGCS. online, 1–34 (2017).
6. Deelman, E., Peterka, T., Altintas, I., Carothers, C.D., Kleese van Dam, K., Moreland, K., Parashar, M., Ramakrishnan, L., Taufer, M., Vetter, J.: The future of scientific workflows. Int J HPC Appl. (2017).
7. De Oliveira, D., Silva, V., Mattoso, M.: How Much Domain Data Should Be in Provenance Databases? In: TaPP. USENIX Association, Edinburgh, Scotland (2015).
8. Davidson, S.B., Freire, J.: Provenance and Scientific Workflows: Challenges and Opportunities. In: SIGMOD. pp. 1345–1350. , New York, NY, USA (2008).
9. F. da Silva, R., Filgueira, R., Pietri, I., Jiang, M., Sakellariou, R., Deelman, E.: A characterization of workflow management systems for extreme-scale applications. FGCS. 75, 228–238 (2017).
10. Bauer A. C., Abbasi H., Ahrens J., Childs H., Geveci B., Klasky S., et al.: In situ methods, infrastructures, and applications on high performance computing platforms. Comp G Forum. 35, 577–597 (2016).
11. Atkinson, M., Gesing, S., Montagnat, J., Taylor, I.: Scientific workflows: Past, present and future. FGCS. 75, 216–227 (2017).
12. Hanzich, M., Rodriguez, J., Gutierrez, N., de la Puente, J., Cela, J.: Using HPC software frameworks for developing BSIT: a geophysical imaging tool. P WCCM ECCM ECFD. 3, 2019–2030 (2014).
13. Lee, K., Paton, N.W., Sakellariou, R., Fernandes, A.A.A.: Utility functions for adaptively executing concurrent workflows. CCPE. 23, 646–666 (2011).
14. Pouya, I., Pronk, S., Lundborg, M., Lindahl, E.: Copernicus, a hybrid dataflow and peer-to-peer scientific computing platform for efficient large-scale ensemble sampling. FGCS. 71, 18–31 (2017).
15. Jain, A., Ong, S.P., Chen, W., Medasani, B., Qu, X., Kocher, M., et al.: FireWorks: a dynamic workflow system designed for high-throughput applications. CCPE. 27, 5037–5059 (2015).
16. Nguyen, H.A., Abramson, D., Kipouros, T., Janke, A., Galloway, G.: WorkWays: interacting with scientific workflows. CCPE. 27, 4377–4397 (2015).
17. Abramson, D., Enticott, C., Altinas, I.: Nimrod/K: Towards massively parallel dynamic grid workflows. In: Supercomputing. pp. 24:1–24:11. IEEE Press, Piscataway, NJ, USA (2008).
18. Gil, Y., Ratnakar, V., Kim, J., Gonzalez-Calero, P., Groth, P., Moody, J., Deelman, E.: Wings: intelligent workflow-based design of computational experiments. IEEE Intellig Sys. 26, 62–72 (2011).
19. Stamatogiannakis, M., Athanasopoulos, E., Bos, H., Groth, P.: PROV2R: Practical Provenance Analysis of Unstructured Processes. ACM Trans Internet Technol. 17, 37:1–37:24 (2017).
20. Bourhis, P., Deutch, D., Moskovitch, Y.: Analyzing data-centric applications: Why, what-if, and how-to. In: ICDE. pp. 779–790 (2016).
21. Silva, B., Netto, M.A.S., Cunha, R.L.F.: JobPruner: A machine learning assistant for exploring parameter spaces in HPC applications. FGCS. 83, 144–157 (2018).
22. Silva, V., Leite, J., Camata, J.J., de Oliveira, D., Coutinho, A.L.G.A., Valduriez, P., Mattoso, M.: Raw data queries during data-intensive parallel workflow execution. FGCS. online, (2017).
23. Ikeda, R., Das Sarma, A., Widom, J.: Logical provenance in data-oriented workflows? In: ICDE. pp. 877–888. , Finland (2013).
24. Camata, J.J., Silva, V., Valduriez, P., Mattoso, M., Coutinho, A.L.G.A.: In situ visualization and data analysis for turbidity currents simulation. Comput. Geosci. 110, 23–31 (2018).
25. Costa, F., Silva, V., de Oliveira, D., Ocaña, K., et al.: Capturing and querying workflow runtime provenance with PROV: a practical approach. In: EDBT/ICDT workshops. pp. 282–289 (2013).
26. Moreau, L., Missier, P.: PROV-DM: The PROV Data Model, https://www.w3.org/TR/prov-dm/.
27. ProvONE provenance model for scientific workflow, http://vcvcomputing.com/provone/provone.html.
28. Oliveira, W., Missier, P., Oliveira, D., Braganholo, V.: Comparing provenance data models for scientific workflows: an analysis of PROV-Wf and ProvOne. In: Brazilian e-Science workshop (2016).
29. PROV-DfA: PROV-DfA GitHub Repository, https://github.com/hpcdb/PROV-DfA.