This article has been accepted for inclusion in a future issue of this journal. Content is final as presented, with the exception of pagination.

An Inference-Based Framework to Manage Data Provenance in Geoscience Applications

Mohammad Rezwanul Huq, Peter M. G. Apers, and Andreas Wombacher, *Member, IEEE*

***Abstract*— Data provenance allows scientists to validate their model as well as to investigate the origin of an unexpected value. Furthermore, it can be used as a replication recipe for output data products. However, capturing provenance requires enormous effort by scientists in terms of time and training. First, they need to design the workflow of the scientific model, i.e., workflow provenance, which requires both time and training. However, in practice, scientists may not document any workflow provenance before the model execution due to the lack of time and training. Second, they need to capture provenance while the model is running, i.e., fine-grained data provenance. Explicit documentation of fine-grained provenance is not feasible because of the massive storage consumption by provenance data in the applications, including those from the geoscience domain where data are continuously arriving and are processed. In this paper, we propose an inference-based framework, which provides both workflow and fine-grained data provenance at a minimal cost in terms of time, training, and disk consumption. Our proposed framework is applicable to any given scientific model, and is capable of handling different model dynamics, such as variation in the processing time as well as input data products arrival pattern. Our evaluation of the framework in a real use case with geospatial data shows that the proposed framework is relevant and suitable for scientists in geoscientific domain.**

***Index Terms*— Data provenance, geoscience applications, hydrology, provenance graph, workflow.**

I. Introduction

A. Data-Intensive Applications and Geoscientific Research

Scientists from many domains, such as physical, geological, environmental, biological etc. facilitate data-intensive e-Science applications to study and better understand these complex systems [1]. In these applications, the data collection contains both in-situ data collected from the field and streaming data sent by sensors. Scientists use this data fitting into their model describing processes in the physical world and get the output, which is used to facilitate either a process control application or a decision support system.

Many of these data-intensive e-Science applications are focusing on geoscientific research. In a geoscientific research, scientists collect geospatial data, i.e., measurements or sensor readings with time and space, from different sources. Later, this data are processed to produce the output, i.e., a data product. A new generation of information infrastructure, known as cyberinfrastructure, is being developed to support the geoscientific research [2]. One of the requirements of this cyberinfrastructure is to trace the creation of the output data products. This path to the origin would be useful in cases of the generation of any imprecise or unexpected output data during the execution of a geoscientific model. To investigate the origin of the unexpected output data, scientists need to debug through their models, which are used for the actual processing.

Furthermore, reproducibility of data products is another major requirement in the geoscientific domain. Reproducibility of data products refers to the ability to produce the same data product using the same set of input data and model parameters irrespective of the model execution time. Maintaining data provenance, also known as lineage, allows scientists to achieve these requirements and thus, leading toward the development of a provenance-aware cyberinfrastructure.

B. Data Provenance

Provenance is defined in many different contexts. One of the earlier definitions was given in the context of geographic information system (GIS). In GIS, data provenance is known as lineage, which explicates the relationship among events and source data in constructing the data product [3]. In the context of database systems, data provenance provides the description of how a data product is achieved through the transformation activities from its input data [4]. In a scientific workflow, data provenance refers to the derivation history of a data product starting from its origin [5]. In the context of the geoscientific domain, geospatial data provenance is defined as the processing history of a geospatial data product [2].

In all contexts, provenance can be defined at different levels of granularity [6]. Fine-grained data provenance is defined at the value-level of a data product, which refers to the determination of how that data product has been created and processed starting from its input values. It helps scientists to trace the value of an output data product. It could be facilitated to have reproducible results as well. On the other hand, coarse-grained or workflow provenance is defined at the more higher level of granularity. It captures association among different activities within the model at design time. Workflow provenance can achieve reproducibility in a few cases where data are collected beforehand, i.e., offline data. In cases of streaming data, workflow provenance itself cannot achieve reproducibility due to the creation of new data products and update of existing data products during the model execution. However, based on the workflow provenance of a

Manuscript received September 30, 2012; revised December 11, 2012; accepted January 25, 2013. This work was supported in part by the Sensor Data Lab Project Funded by the University of Twente.

The authors are with the Department of Computer Science, Database Group, University of Twente, Enschede 7522NH, The Netherlands (e-mail: m.r.huq@utwente.nl; p.m.g.apers@utwente.nl; a.wombacher@utwente.nl).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TGRS.2013.2247769

0196-2892/$31.00 © 2013 IEEE

This article has been accepted for inclusion in a future issue of this journal. Content is final as presented, with the exception of pagination.

model, we can infer fine-grained data provenance which can significantly reduce storage overhead for provenance data. Therefore, a framework integrating both workflow and fine-grained data provenance will be proven beneficial to scientists using provenance data.

C. Goal of This Research

We aim to develop a framework managing both workflow and fine-grained data provenance for data-intensive, geoscientific applications. To accomplish such a framework, we identify three key design factors. First, the framework would be generic, i.e., applicable to any given model. The biggest challenges to make the framework generic in nature is to address different types of developing approach, i.e., with or without facilitating any specific tools, as well as to address different types of representation of model's structure (e.g., data-flow or control-flow) [7]. Second, the framework should be storage-efficient, i.e., manage provenance data at lower disk consumption. To accomplish this feature, instead of documenting fine-grained data provenance explicitly, inference-based methods could be an alternative solution. Finally, the framework should be self-adaptable to cope with any given scientific model and the model dynamics, such as processing delay, data arrival pattern etc. The self-adaptability of the framework decides to apply an appropriate provenance inference method based on the aforesaid parameters to build provenance traces. Accomplishing a framework with these properties requires us to closely examine the complete problem domain, i.e., entities involved with geoscientific applications.

D. Complete Problem Domain

In the following, the problem space is described, which is addressed by the proposed framework. The problem space can be characterized into two phases: design phase and execution phase. Fig. 1 uses rectangles and round-shaped boxes to represent different entities pertinent to a geoscientific model and the corresponding example based on their characteristics, respectively. The entities defined during the design phase of a scientific model are: 1) the scientific model itself and 2) different activities within the model. These two entities are represented by the top two rectangles in Fig. 1. The entities involved during the execution phase of a scientific model are represented by the bottom two rectangles shown in Fig. 1. Each activity defined at the design phase instantiates a corresponding processing element during the execution of a model and these processing elements process incoming data products and produce output data products. We discuss different characteristics of these entities below.

1) Design Phase Characteristics: During the design phase, scientists define the model which is based on different activities, i.e., atomic units of work performed as a whole [8]. In case the scientific model is specified in a provenance-aware platform, the provenance information is automatically acquired. Examples of platforms, where provenance awareness has been considered, are e-Science workflow engines, such as Kepler [9], Karma2 [10], Taverna [11], VisTrails [12], stream processing and complex event processing engines, such as

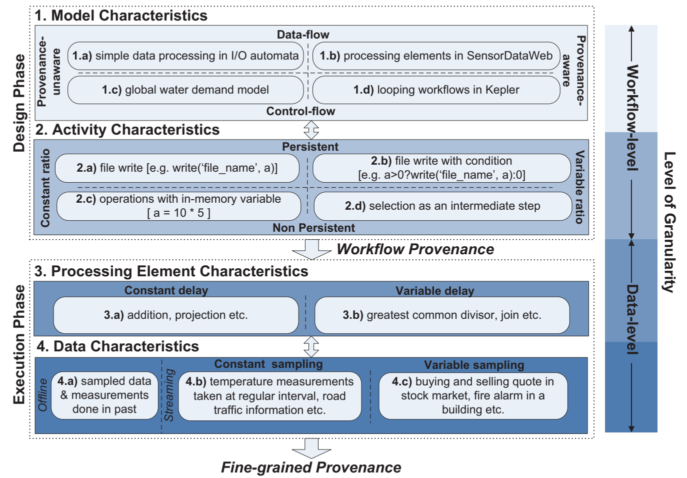


Fig. 1. Complete problem domain showing different characteristics of a scientific model at design and execution time.

SensorDataWeb,[1] STREAM [13], Aurora [14], Borealis [15], or Esper.[2] Provenance has been considered in these platforms because they could be used to model provenance-aware applications.

In case the scientific model is specified in a provenance-unaware platform, the workflow provenance must be maintained manually by the user. This requires training of the user and a significant effort in manually acquiring provenance information. Examples of provenance-unaware platforms are general purpose programming and scripting languages, such as Python,[3] general purpose data manipulation tools, such as Microsoft Excel,[4] R,[5] or MATLAB.[6]

The second dimension of classifying scientific models is based on the underlying coordination approach of the model. In control-flow coordination, the execution of an activity depends on the successful completion of the preceding activity. This paradigm is used in many programming languages that a statement can only be executed after the previous statement has been completed. It also applies to many workflow models and logical formulations. As a contrast, in data-flow coordination, the execution of an activity depends on the availability of data and that in turn, can potentially produces data again, which may trigger the execution of other activities. This paradigm is used in stream processing and complex event processing environments as well as in the models used in distributed systems research, such as I/O automata [16].

The next entity in the design phase is activities. Several activities comprise a scientific model. There are two important characteristics of an activity, which need to be documented to help scientists finding and understanding the origin of a data product during execution phase. One of them is input–output ratio. The input–output ratio [17] refers to the ratio between the number of contributing input data products to the number of produced output data products. There are many

[1] Available at https://sourceforge.net/projects/sensordataweb/.
[2] Available at http://esper.codehaus.org/.
[3] Available at http://www.python.org/.
[4] Available at http://office.microsoft.com/en-us/excel/.
[5] Available at http://www.r-project.org/.
[6] Available at http://www.mathworks.nl.

This article has been accepted for inclusion in a future issue of this journal. Content is final as presented, with the exception of pagination.

activities where this ratio remains constant during execution phase, such as arithmetic operations without any condition, projections, aggregate operations in a database etc. We refer to these activities as constant ratio activities. On the other hand, there are a few activities which do not keep the input–output ratio constant during execution phase, such as a typical selection operation in a database etc. These are referred to as variable ratio activities.

The other important characteristic is about the persistence of the output data product, referred to as IsPersistent. The IsPersistent characteristic describes whether the data product produced by an activity is stored persistently or not. If the data product is persistent, it can be used to infer fine-grained data provenance. As an example, in general purpose programming languages, writing to a file results into persistent data. Documenting the input–output ratio and the IsPersistent characteristics and potentially the other characteristics of the activities during the design phase explicated in the workflow provenance helps scientists to understand the origin of a data product during execution phase.

The documented characteristics and the relationship between activities during the design phase result into the workflow provenance of the scientific model. While the workflow provenance is acquired automatically in a provenance-aware platform, this must be done manually in a provenance-unaware platform. However, there is a high demand in the scientific community to capture workflow provenance automatically in a provenance-unaware platform like a scripting environment [18]. To accomplish this, the challenge is to transform the data- and control-flow aspects of a scripting language into data dependences between activities, i.e., workflow provenance, by interpreting and analyzing the code. That is to transform a control-flow statement (e.g., function call) into an activity or a group of activities, which only exhibits data dependences.

Please note that in different scientific models, activities have different granularities ranging from complex operations to a single arithmetic operation. While the granularity of the activities does not influence the provenance acquisition, it is influencing the complexity of the provenance graph and the interpretation by the user.

2) Execution Phase Characteristics: The entities involved during the execution phase are: 1) processing elements and 2) data. These entities and their characteristics are explicated using the bottom two rectangles in Fig. 1. An activity defined in the design phase is transformed into a corresponding processing element during the execution phase. Processing elements have variations in their processing delay, i.e., amount of time required to manipulate input data products. As an example, processes performing addition or projections have constant processing delays, referred to as constant delay processing elements. Alternatively, executing some processing elements, such as performing a join in a database or calculating the greatest common divisor, require different amount of time at each execution. These are referred to as variable delay processing elements.

Independent of processing elements characteristics, the contributing data also exhibit its own characteristics. Data might arrive continuously (e.g., streaming data) or can be collected before the execution begins (e.g., offline data). Streaming data might have different data arrival patterns. Data arriving at regular intervals are referred to as constant sampling data (e.g., temperature measurements sent at regular intervals). On the other hand, data might also arrive at an irregular interval, such as buying and selling quotes on an instrument in a stock market. These are referred to as variable sampling data.

The relationship between data and processing elements during the execution phase explicates the fine-grained data provenance of a scientific model [6]. Existing work documents fine-grained data provenance explicitly in a database [19], [20]. However, these mechanisms require a considerable amount of storage to maintain fine-grained data provenance especially in a data streaming scenario where a single incoming data product may contribute to produce multiple output data products. Sometimes, the size of provenance data becomes a multiple of the actual data. Since provenance data are just metadata and less often used by the end users, the explicit documentation of fine-grained provenance seems to be infeasible and too expensive [21]. One of the potential solutions to overcome this problem is to infer fine-grained data provenance based on the given workflow provenance. Therefore, inferring the fine-grained data provenance can make the complete framework storage-efficient.

However, developing an inference-based framework to manage both workflow and fine-grained data provenance requires attention to the underlying platform along with the model dynamics, including processing element and data characteristics. The inference mechanisms should take variation in the used platform, processing delay, and data arrival pattern into consideration to infer highly accurate provenance information. To accomplish that, self-adaptability of the framework is required, which can decide when and how to execute the most appropriate inference-based methods based on a given scientific model and its associated data products.

E. Our Contribution

In this paper, we describe an inference-based framework to manage both workflow and fine-grained data provenance for geoscientific applications. Our proposed framework is applicable to any given model specified in either a provenance-aware or a provenance-unaware platform using data-flow or control-flow-oriented structure, which conforms to its generic nature. To accomplish that, we propose a technique to build the workflow provenance automatically based on a given script. This overcomes the difficulties with collecting workflow provenance automatically for a model developed on top of a provenance-unaware platform, such as a scripting language. Since there are many programming and scripting languages and each has its own set of programming constructs and syntax, we showcase our approach using Python. Python is widely used to handle spatial and temporal data in the scientific community as well as in commercial products, such as ArcGIS,[7] which has inspired us to make this choice.

Our proposed framework is also capable of managing fine-grained data provenance in a storage-efficient way. We discuss

[7] Available at http://www.esri.com/software/arcgis.

the applicability of several existing provenance inference methods [17], [22], [23] in this paper, which can infer provenance based on the given workflow provenance of the model and the timestamps associated with data products.

To achieve a self-adaptable framework, we build a decision tree not only to control when both workflow and fine-grained provenance inference methods are executed but also to decide dynamically per processing element which fine-grained inference method to use for the inference based on the observed model dynamics, i.e., data arrival pattern, processing delay etc. The inference of fine-grained data provenance over the complete scientific model allows the users to debug the model by specifying the space and time of interest.

These contributions are made accessible to users by developing a tool known as ProvenanceCurious, which visualizes provenance information as a graph. Using our framework, scientists can utilize both workflow and fine-grained data provenance with minimal effort in terms of time, training, and storage cost. We have also evaluated the proposed framework based on a case-study, involving a model for estimating the global water demand [24]. This model includes geospatial data, i.e., raster maps with timestamp. Our evaluation demonstrates the applicability and suitability of using the proposed framework in geoscientific domains. Furthermore, we briefly discuss the performance of the existing provenance inference methods in terms of storage consumption and accuracy using both real and synthetic dataset. The proposed inference-based framework provides scientists sufficient information to investigate the unexpected behavior of a scientific model.

II. Use Case: Estimating Global Water Demand

To illustrate the problem description in Section I-D, the use case for estimating the global water demand based on the scientific model reported in [24] is introduced. The model used in this use case is developed using a provenance-unaware platform, i.e., Python language, and the model manipulates offline data.

A. Background of the Model

Freshwater is one of the most important resources for various human activities and food production. During the past decades, use of water has been increased rapidly, yet available freshwater resources are finite. Therefore, estimating water demand and availability on a global level is necessary to assess the current situation as well as to make policies for the future. In this use case, we focus on the script that estimates the total water demand from the year 1960 to 2000 at a monthly resolution by facilitating the collected geospatial data from different sources.

B. Model Inputs

Source data are collected from different existing datasets. Irrigated areas are prescribed by the MIRCA2000 dataset [25] and the FAOSTAT database.[8] Crop factors, growing season lengths, and rooting depth are obtained from GCWM [26]. The

[8]Available at http://faostat.fao.org/.

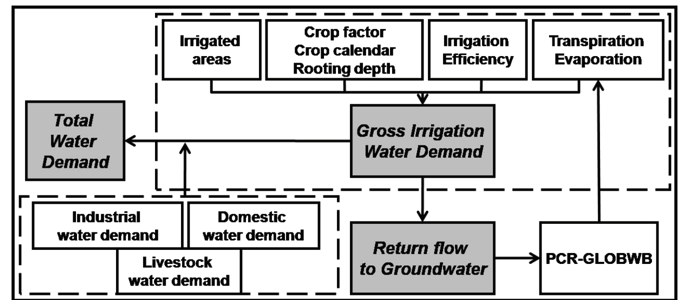


Fig. 2. Different types of data and their dependency in the use case.

irrigated areas are representative for the period 1960–2000 at a yearly temporal resolution, i.e., remains constant over each year, while the crop-related datasets are representative for the year 2000 at a monthly temporal resolution. A map of country-specific irrigation efficiency factors is also obtained from [27]. In addition, daily potential and actual bare soil evaporation and transpiration are prescribed from the simulation results from the global hydrological and water resources model PCR-GLOBWB [28]. Fig. 2 shows the input and output data and the dependences between them. The white boxes are input data collected from various sources and the shaded boxes represent output data of this model. The edges represent data dependences from one to another.

C. Model Activities

The model begins with reading the annual and monthly input maps described above. First, using irrigated areas, crop factors, growing season lengths, and potential transpiration, we calculate potential crop transpiration. Then, we calculate actual crop transpiration and determine the difference between potential and actual crop transpiration. In addition, we compute the difference between potential and actual bare soil evaporation for the top soil layer. Net irrigation water demand thus equals the sum of the differences between the potential and actual crop transpiration and between the potential and actual bare soil evaporation. However, much of this water is lost to evaporation and percolation during the transport and application. Therefore, we calculate irrigation loss and add this to the net irrigation demand. At last, we use country-specific irrigation efficiency factors and multiply these with the net irrigation water demand to yield gross irrigation water demand.

The estimated gross irrigation water demand is then added to other sectoral water demands, i.e., industrial, domestic, and livestock water demand that are directly read from maps. Furthermore, we use gross irrigation water demand to calculate return flow to groundwater.

D. Model Outputs

Finally, the resulted total water demand, gross irrigation water demand, and irrigation return flow are reported as PCRaster maps (shaded boxes in Fig. 2) containing geospatial data for each year from 1960 to 2000 at a monthly temporal resolution.

This article has been accepted for inclusion in a future issue of this journal. Content is final as presented, with the exception of pagination.

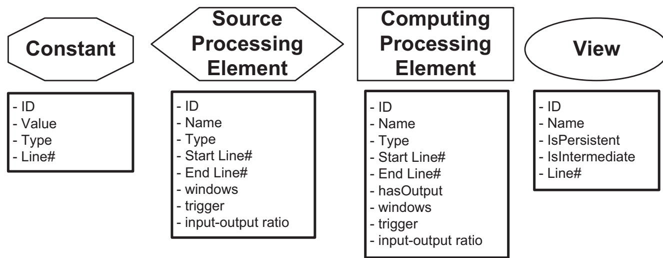


Fig. 3. Different nodes and their properties in a provenance graph.

III. Workflow Provenance

A. *Workflow Provenance Model*

The core concept of the proposed framework is the workflow provenance model. Workflow provenance is represented as a graph referred to as workflow provenance graph. A provenance graph G_p is a set of (V, E) where V denotes the set of vertices or nodes and E denotes the set of directed edges. We introduce a graph model to distinguish different types of nodes. In our graph model, there are four different types of nodes. These are as follows.

1) *Constant*: Represents any constant value taking part in an operation.
2) *Source Processing Element*: Represents any operation that either assigns a constant or reads data from the disk.
3) *Computing Processing Element*: Represents any operation that either computes a value based on its parameters or writes data into the disk.
4) *View*: Represents either any variable defined in the script or an intermediate result generated by a processing element.

A directed edge connecting two nodes represents the data dependence in a workflow provenance graph. In the workflow provenance model, every source and computing processing element generate a view. Further, a view or constant node can be used as an input for multiple source and computing processing elements.

Each type of nodes has different properties. Fig. 3 shows the graphical representation of these nodes and their properties. A constant node has an id starting with "C," a value, the type of the value (e.g., integer, string etc.) and a line# referring to the line number in the code where it is defined. All nodes have this line# property. Since source and computing processing elements could be defined over multiple lines, they have start and end line#.

A view node has an id prefixed with "V" and a name (variable name). It also has two important boolean properties: 1) IsPersistent and 2) IsIntermediate. When IsPersistent = true, it means that the variable which corresponds to this view is read from the disk or is written into the disk and hence, persistent. Otherwise, the view is not persistent and thus IsPersistent becomes false. The property IsIntermediate is true when the view is produced by a processing element and contains an intermediate result. Otherwise, IsIntermediate becomes false and it indicates that the view is created because of defining the corresponding variable in the script.

The set of properties of both source and computing processing elements are almost similar except one property, hasOutput. This property belongs to a computing processing element, which indicates whether a produced result is persistent, i.e., written into the disk, or not. Since source processing elements only read data from the disk, hasOutput is not applicable for a source processing node. Moreover, both source and computing processing elements have an id prefixed with "SP" and "P," respectively, a name and type of operation (e.g., binary, function call etc.). The other properties of both source and computing processing elements shown in Fig. 3 are: 1) windows; 2) trigger; and 3) input–output ratio.

1) Windows: A window specifies a subset of data products used by an activity to produce an output data product. Therefore, a window with a predefined size is applied over the input data products, i.e., views, to limit the number of data products to be considered by the processing element. Windows could be defined based on the number of tuples, i.e., tuple-based window, or time units, i.e., time-based window.
2) Trigger: A source or computing processing element is repeatedly executed after elapsing a predefined interval, also known as trigger period. The interval is defined based on the number of tuples, i.e., tuple-based trigger, or time units, i.e., time-based trigger.
3) Input–Output Ratio: It refers to the ratio of the number of the data products contributed in a processing element to the number of the data product produced by the same processing element during the execution phase. As for example, an aggregate operation considers all input data products to produce an output data product. Therefore, the input–output ratio for the processing element representing aggregates is $n : 1$ where n is the number of input data products in the window.

This set of properties are quite similar to the process provenance reported in [10]. The value of these three properties is inferred by analyzing the control- and data-flow of the given script. Since in a provenance-unaware platform manual preparation of workflow provenance is time consuming, maintaining workflow provenance automatically by inference is a much needed initiative that can save significant amount of time. Furthermore, this inference-based workflow provenance management can easily cope with the change in the scripts used in the model.

B. *Workflow Execution Model*

After collecting the workflow provenance automatically, the workflow is executed based on the execution model described in [29]. In cases of streaming scenarios, data are continuously propagated through the workflow. The execution never stops as long as new data products arrive and the arrival of new data triggers an activity, which is receiving the data. Different activities are loaded into the execution engine, where each activity is performed based on a time or tuple-based trigger. For a time-based trigger, at every point in time where the trigger predicate is interpreted as valid, the activity is fired and output data products are produced. For a tuple-based trigger,

This article has been accepted for inclusion in a future issue of this journal. Content is final as presented, with the exception of pagination.

it is more difficult to predict when the next trigger will be enabled. Therefore, the particular activity continuously checks whether the set of new data products observed at a specific time is sufficient to validate the trigger predicate as true. If this is the case then the activity is executed and output data products are generated.

C. Provenance Representation and Sharing

It is important to choose an appropriate provenance representation model to enable interoperability for sharing provenance data. To achieve interoperability, the PROV data model, known as PROV-DM, could be facilitated to represent provenance information [30]. PROV-DM is a generic data model for provenance that allows domain- and application-specific representations of provenance to be translated into such a data model and then allowing interchange between systems. Therefore, PROV-DM is domain and application agnostic and heterogeneous systems can export their native provenance into such a core data model.

In GIS, there is a standard known as ISO 19115:2003[9] which defines the schema required for describing geographic information and services. This metadata can be translated into Geography Markup Language[10] (GML) which is the XML grammar defined by the open geospatial consortium (OGC) to express geographical features. GML serves as a modeling language for geographic systems as well as an open interchange format for geographic transactions on the Internet. Yue *et al.* [39] proposed to facilitate XML encoding to represent provenance data for better interoperability. In our proposed framework, since provenance data are represented as a graph, we facilitate GraphML[11] which is a XML-based file format for exchanging graph structure data. GraphML is supported by most of the graph editing tools such as yEd,[12] Gephi[13] etc. Therefore, provenance graphs generated by the proposed framework can be shared easily throughout the scientific community.

IV. Workflow Provenance Inference in a Provenance-Unaware Platform

A. Overview of the Approach

The proposed workflow provenance inference mechanism can infer workflow provenance information by analyzing a Python script. A typical Python script might be comprised of assignment, arithmetic operation, user-defined function call, conditional branching, looping etc. Activities, such as assignment, arithmetic operation are purely based on data-flow coordination where availability of data triggers the next activity. However, other activities, such as user-defined function call, conditional branching, looping etc. are implemented by using control-flow based coordination and result into control dependences between activities. Since data provenance identifies the data dependences between activities, control dependences must be transformed into data dependences to infer the workflow provenance.

A control-flow maintains the dependence in such a way that an activity is started only after the preceding activity has been completed. As a consequence, variables defined or updated in an activity are accessible after the activity has been completed. Therefore, control dependence can be represented as data dependence by versioning the variables. In particular, a new version of a variable is created after its modification by an activity. The control dependence determines which version of a variable is used by a read operation of an activity on the variable.

We start the inference mechanism by parsing a given Python script based on a combined grammar, containing parser and lexer rules. After parsing the script, it returns an abstract syntax tree (AST) for the given Python script. Then, we traverse through this AST based on a tree grammar and for each node in the AST, an object of the appropriate class based on the object model of Python is created. Having obtained all objects, we can build the initial provenance graph maintaining the syntactic relationship between these objects, including control-flow based coordination.

Since the initial provenance graph preserves the control-flow based coordination and contains some extra nodes due to the syntactic sugar of Python, it needs to be transformed in a form where the graph exhibits data dependences only and becomes more compact. Therefore, we introduce a set of re-write rules to transform the initial provenance graph into the workflow provenance graph. A re-write rule has two parts: left-hand side (LHS) and right-hand side (RHS). Once a rule is defined and is executed, it searches for an isomorphic sub-graph equivalent to the sub-graph pattern mentioned in the LHS of the rule. If the pattern is found, it is replaced by the sub-graph in the RHS of the rule.

The first set of rules are used to transform the control-flows in the initial provenance graph into the data-flows and are referred to as flow transformation rules. Furthermore, we apply another set of rules to propagate the persistent property of a view to the next view where applicable as well as to identify the processing element, which produces output data. One of the rules in this set discards intermediate nodes. We name this set of rules as graph maintenance rules. Finally, we apply model modification rules to reduce the number of nodes further by integrating the view and constant nodes with processing element nodes. Each re-write rule in these sets is executed one after another according to the aforesaid sequence. Execution of a particular rule is stopped and switched to the next rule if there is no other isomorphic subgraph found in the initial provenance graph, equivalent to the sub-graph pattern mentioned in LHS of that particular rule. After applying all these re-write rules, we get the workflow provenance graph.

We have used an off-the-shelf grammar[14] as a starting point and extend it according to our requirements to obtain the AST. In this section, we focus on the mechanism of creating an initial graph and building a workflow provenance graph from the initial graph based on the set of graph re-write rules.

[9] Available at http://www.iso.org/iso/catalogue_detail.htm?csnumber=26020.
[10] Available at http://www.opengeospatial.org/standards/gml.
[11] Available at http://graphml.graphdrawing.org/.
[12] Available at http://www.yworks.com/en/products_yed_about.html.
[13] Available at https://gephi.org/.
[14] Available at http://www.antlr.org/grammar/1200715779785/python.g.

This article has been accepted for inclusion in a future issue of this journal. Content is final as presented, with the exception of pagination.

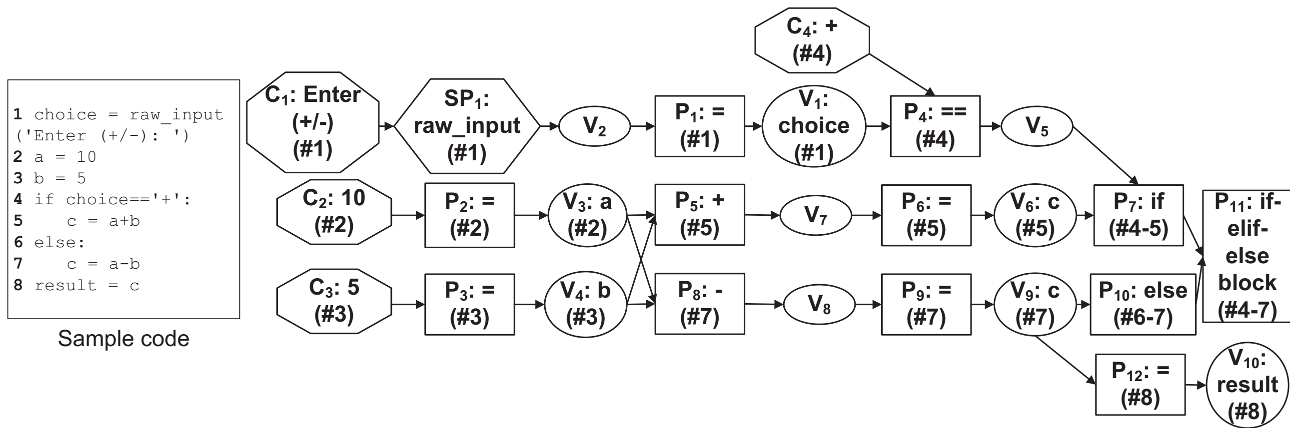


Fig. 4. Example of the initial provenance graph.

B. Creating Initial Provenance Graph: An Example

Building an initial provenance graph depends on the created objects based on the object model of Python and their syntactic relationship to each other. We generate the initial provenance graph by facilitating attributed graph grammar (AGG),[15] which is a graph writing engine.

Fig. 4 shows a sample script and the initial provenance graph of the given script. In this script, first, the user is asked to enter either "+" or "−," which is then assigned into the variable choice in line 1. Based on the value of choice, the script calculates either addition or subtraction of two variables a and b, which are assigned with the value 10 and 5, respectively, in lines 2 and 3. It then assigns the result into the variable c. Afterward, the variable result is assigned with the value hold by c in line 8.

In Fig. 4, the source processing element node SP_1 represents the raw_input method, which allows the user to enter either "+" or "−" represented by the constant node C_1 and assigns the value into choice denoted by the view node V_1. For each method used in a given script, the user has to provide a few information beforehand, such as: whether a method reads data from disk (true/false) and whether a method writes data into disk (true/false) to make a distinction between source and computing processing elements.

The nodes P_2 and P_3 in Fig. 4 represent the assignment operations, which assign the value 10 and 5 denoted by C_2 and C_3 into the variables a and b represented by nodes V_3 and V_4 according to lines 2 and 3, respectively. Later, these views, V_3 and V_4 participate to the addition and subtraction operation represented by nodes P_5 and P_8, respectively. The output of these two nodes is then assigned to two different versions of the same variable c, denoted by V_6 and V_9, respectively, due to the conditional branching defined within lines 4–7.

To represent the control-flow resulting from the aforesaid conditional branching, we create the node P_{11} which holds the block of statements in lines 4–7. Moreover, in the initial provenance graph shown in Fig. 4, there are two nodes created for if and else branch found in the code, denoted by the nodes P_7 and P_{10}, respectively. Both of these nodes are connected to P_{11} as they are parallel branches of the same conditional block. They get connected from the views created within their scope (e.g., V_6 to P_7, V_9 to P_{10}). Moreover, these nodes are also connected from the view that holds true or false, representing the status of the given condition in a particular branch (e.g., V_5 is connected to P_7). Eventually, the node P_{12} assigns the value of c into the new variable result represented by V_{10}.

The initial provenance graph is developed in such a way so that it can transform the implicit control-flow between statements that define the order of execution between processing elements into data dependences. However, the initial provenance graph shown in Fig. 4 exhibits explicit control-flow coordination due to the conditional branching in the given script. In the next section, we discuss the set of re-write rules, which transform control dependences into data dependences.

C. Flow Transformation Re-Write Rules

One of the biggest challenges to infer workflow provenance directly from scripts is to transform the control dependences into data dependences, i.e., execution of a processing element only depends on the availability of data, not on the execution of the preceding processing element. To transform control dependences into data dependences, we define a set of rules, one for each type of control-flow statements. We consider three types of statements, involving control dependences. These are: 1) conditional branching (e.g., if-elif-else); 2) looping (e.g., for); and 3) user-defined function/subroutine call (e.g., passing parameters to a defined function and assigned the returned value into a variable).

1) Conditional Branching: Conditional branching refers to the execution of a set of statements only if some condition is met. A conditional branching statement exhibits control-flow based coordination and is translated into data dependence by correlating a variable read of an activity in a conditional branch to the latest version of that variable available before the conditional branching language construct. All conditional branches are represented as parallel data dependences each containing an additional activity with variable ratio, i.e., selectively forwarding the data product based on the condition. Then, all parallel branches are condensed into a single data dependence again using a union activity.

[15] Available at http://user.cs.tu-berlin.de/ gragra/agg/.

This article has been accepted for inclusion in a future issue of this journal. Content is final as presented, with the exception of pagination.

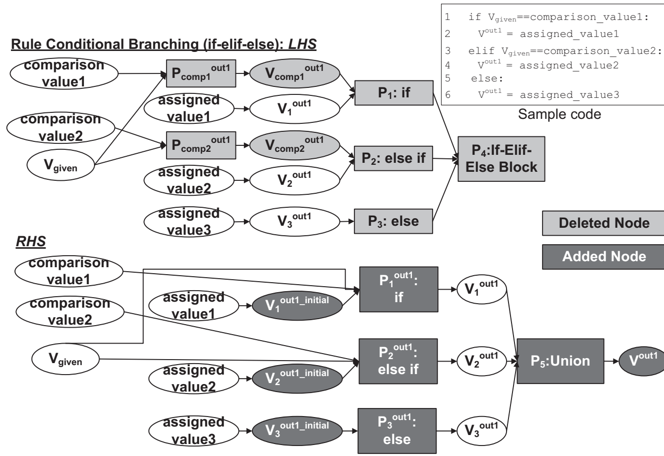


Fig. 5. Re-write rule for conditional branching.

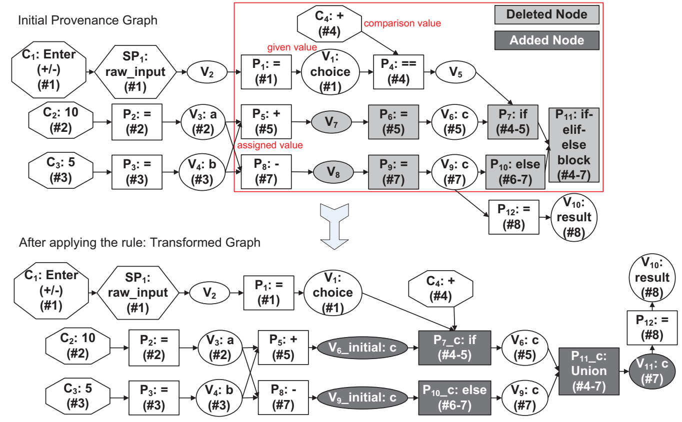


Fig. 6. After applying the re-write rule for conditional branching on the initial provenance graph.

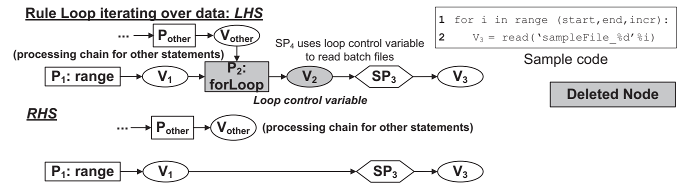


Fig. 7. Re-write rule for a loop that iterates over data products.

LHS of Fig. 5 shows the sub-graph pattern that could be found in the initial provenance graph if a conditional branching is defined in a given script. In the initial provenance graph, a computing processing element (e.g., P_1, P_2, and P_3) is created for each of these conditional branches. Each of these processing elements has two parts: 1) conditional part and 2) activity part if the condition is met, connecting toward itself. For P_1, the conditional part is originated from the node comparison_value1 and the view V_{given}, i.e., if (V_{given} == comparsion_value1). The activity part is originated from the node assigned_value1 that represents an assignment into the variable V^{out1}, i.e., (V^{out1} == assigned_value1), if the condition is met. Since V^{out1} may hold different values depending on the condition that is satisfied, we denote these different versions of out1 as the view nodes $V_1{}^{\text{out1}}, \ldots, V_n{}^{\text{out1}}$ where n is the total number of conditional branches in the current scope.

To transform the control dependences into data dependences in a conditional branch statement, we use the concept introduced in a program representation graph [31]. In a program representation graph, after every conditional branch one extra node is added to represent the output variable to follow static single assignment forms [32]. Therefore, in the RHS of Fig. 5, we replace the nodes $V_1{}^{\text{out1}}$, $V_2{}^{\text{out1}}$, and $V_3{}^{\text{out1}}$ with the nodes $V_1{}^{\text{out1_initial}}$, $V_2{}^{\text{out1_initial}}$, and $V_3{}^{\text{out1_initial}}$, representing the potential value to be assigned if that particular branch satisfies the condition. After checking the condition, the value could be assigned to any of these nodes $V_1{}^{\text{out1}}$, $V_2{}^{\text{out1}}$, and $V_3{}^{\text{out1}}$. Therefore, they have been placed after P_1, P_2, and P_3, respectively. Furthermore, several activities could be carried out if a particular condition is met. Therefore, each processing element is decomposed into multiple instances where each instance of the same processing element handles exactly one activity and produces the corresponding output. In the RHS of Fig. 5, processing elements $P_1{}^{\text{out1}}$, $P_2{}^{\text{out1}}$, and $P_3{}^{\text{out1}}$ represent the instance of P_1, P_2, and P_3, respectively, created for handling the assignment activity of the variable out1. Depending on the condition, only one of these nodes: $V_1{}^{\text{out1}}$, $V_2{}^{\text{out1}}$, and $V_3{}^{\text{out1}}$, actually holds the value of out1. Finally, we add an union processing element to capture the data available in one of these nodes and produce the view V^{out1}, representing the value assigned into out1 variable. RHS of Fig. 5 shows the pattern after the transformation. The light and dark shaded nodes in LHS and RHS represent the deleted and added nodes, respectively.

Fig. 6 shows the transformation of the initial provenance graph (see Fig. 4) after applying the rule for conditional branching. The sub-graph pattern mentioned in the LHS of the rule for conditional branching is represented by the nodes surrounded by the rectangle in the initial provenance graph. After applying the aforesaid rule, the transformed graph is shown in the bottom part in Fig. 6. The newly created processing elements in the transformed graph have trigger $= 1$ and $1:1$ input–output ratio. All associated views have window size $= 1$.

2) Looping Constructs: In any programming language, loops are used for different purposes. We identify two major operations of looping constructs. First, loops can be used for iterating over input data products only. As an example, the usage of a loop to iterate over several input files falls into this category. The other usage of loop is to manipulate input data products to produce new output data products. As an example, usage of a loop to produce running sum over a defined range of data tuples, executing at a fixed interval. In the former case, the default window size and trigger rate is 1. However, in the later that manipulates data products, the window and trigger depends on the boundary and increment value used in defining a loop. Therefore, in this case, the control dependence is translated into data dependence by inferring the window size and trigger predicate of the activities defined within the loop.

LHS of Fig. 7 shows the sub-graph pattern that could be found in the initial provenance graph if the given script has a looping construct iterating over data products. Based on the sample code shown in Fig. 7, each iteration of the

This article has been accepted for inclusion in a future issue of this journal. Content is final as presented, with the exception of pagination.

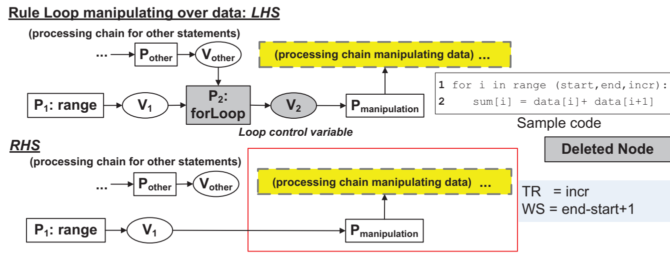


Fig. 8. Re-write rule for a loop that manipulates data products.

loop reads an input file (e.g., sampleFile_1, sampleFile_2 etc.) based on the value of the loop control variable, i. The processing element P_2 represents the defined loop and it takes the range parameters as input and produces the view node V_2, representing the loop control variable. Later, if the loop control variable, i represented by V_2, is used only in an activity that reads source data (SP_3), we conclude that the loop is used to iterate over data products, not to manipulate data products. In this case, the processing element referring to the loop, P_2, and the corresponding loop control variable V_2 are eliminated from the initial workflow provenance graph to transform the control dependences into the data dependences. RHS of Fig. 7 shows the data dependences after the transformation where the execution of SP_3 only depends on the data in V_1. In this case, the window size and trigger rate of SP_3 is 1 and the input–output ratio is 1 : 1.

However, in the other case, when the looping constructs are used to manipulate input data products to produce output data products, the transformation of control dependences into data dependences is accomplished by inferring the window size and trigger rate of the manipulating processing elements. LHS of Fig. 8 shows the sub-graph pattern when a loop is used to produce a new output data product (sum[i]) by adding two input data products (data[i], data[i+1]). In Fig. 8, P_2 represents the loop and V_2 is the view node created for representing the loop control variable, i. If V_2 participates in a manipulating processing element $P_{\text{manipulation}}$, which manipulates input data products and results into a new output data product as in this case, we can infer the window size and trigger rate of the manipulating processing elements from the given range parameters defining the loop. V_1 holds these parameters (start, end, incr). The first two specifies the boundary of the loop control variable, i, represented by V_2. Therefore, the window size of the manipulating processing element $P_{\text{manipulation}}$ and the successive processing elements in the chain is end-start+1. The last parameter incr refers to the increment of the i and therefore, it is the trigger rate of the manipulating processing elements and other successive processing elements in the chain. The processing elements enclosed within the rectangle in RHS of Fig. 8 have the inferred window size and triggers, which are documented. The input–output ratio of these processing elements remains same as it was.

3) User-Defined Function/Subroutine Call: Function or subroutine call executes a set of statements defined in the body of the function, after which the flow of control usually returns to the activity which calls the particular function. Since the successful execution of the caller activity and other activities to be executed after the caller activity depends on

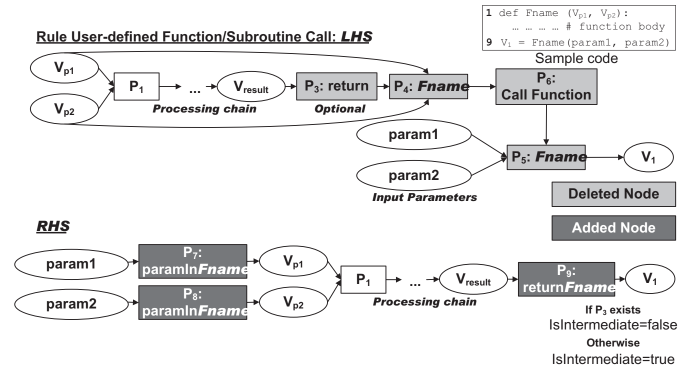


Fig. 9. Re-write rule for a user-defined function/subroutine call.

the successful completion of the activities defined within the function, function call exhibits control-flow based coordination between activities.

To transform the control dependences into data dependences in a user-defined function call, we replicate the nodes within the function body into the place where the caller activity calls the function. LHS of Fig. 9 shows the sub-graph pattern for a function call in the initial graph based on the sample code. The user-defined function Fname is defined and later in the code it is called. The processing element P_5 represents the caller activity and it takes parameters represented as param1 and param2 and calls the function which Fname is defined by the processing element P_4. P_6 connects the caller P_5 to the function body hold by P_4.

For the transformation, we introduce two specific activities:paramIn and return. The paramIn activities take parameters from the caller P_5 as input and then connects them to the parameters mentioned in the function definition hold by P_4. Then, the processing chain defined within P_4 is replicated. The return activity takes the returned value from the function if there is any and assigns the value into the view V_1 with IsIntermediate = false. Otherwise, it assigns an intermediate value into V_1 (IsIntermediate = true). RHS of Fig. 9 shows the pattern after the transformation.

In this case, the trigger rate is 1 for all the processing elements and the window size is also 1. However, the input–output ratio of the processing elements defined in the function must be given by users at the beginning of the model execution because the input–output ratio cannot be inferred for a user-defined function.

D. Graph Maintenance Re-Write Rules

This set of re-write rules are defined to ensure the propagation of persistence of views from one to another as well as to discard unnecessary intermediate views followed by the assignment processing element. Furthermore, one of the rules in this set helps scientists to identify the computing processing element, which generates persistent output data products.

Fig. 10 shows three graph maintenance rules. The first rule propagates the persistence of a view to the next one if some conditions hold. This rule ensures that if scientists use Python methods, such as read to read input data products from persistent storage and to assign this data into a variable,

This article has been accepted for inclusion in a future issue of this journal. Content is final as presented, with the exception of pagination.

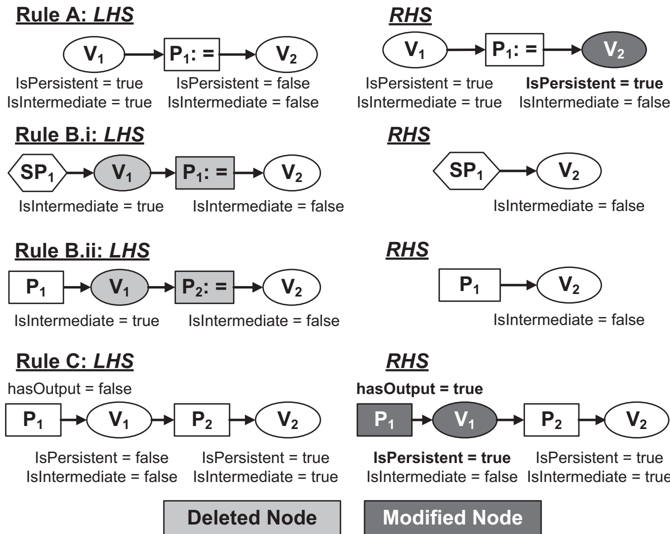


Fig. 10. Re-write rules for graph maintenance.

the corresponding view created for the variable will also be persistent (IsPersistent = true). Rule A in Fig. 10 shows the sub-graph pattern for such an activity. The intermediate view V_1 is produced by the read method and contains persistent data. Later, the persistent data hold by V_1 is assigned into a variable represented by V_2 through processing element P_1. In this case, the persistence of view V_1 is propagated toward view V_2 and V_2 becomes also persistent (IsPersistent=true). RHS of rule A shows the changed property of V_2.

Rule *B* minimizes the size of the workflow provenance graph. It deletes all intermediate views (IsIntermediate = true) and subsequent assignment process nodes (name = "=") if they are followed by a view representing a variable defined in the script, i.e., (IsIntermediate = false). It has two variants depending on the type of the node, which produces the intermediate view shown in the LHS of Fig. 10.B.i and 10.B.ii. Executing these rules, discard the light-shaded nodes from the initial graph and makes a connection between SP_1 and V_2 as well as between P_1 and V_2 for rules B.i and B.ii, respectively.

Rule *C* identifies the computing processing element generating a persistent result, i.e., the result that is written into the disk. In Python, there are a few methods, such as write, report, which writes data into the disk. However, these methods do not compute the data rather they write the data produced by another processing element. Therefore, the processing element which produces the data that is written into the disk later, is the computing processing element generating persistent output. LHS of Fig. 10.C shows the sub-graph pattern for the aforesaid activities. P_2 is the processing element representing the method, such as write or report and it generates a view V_2 which refers to the data written into the disk. Before, P_2 has taken V_1 as input and V_1 is nonintermediate view, it means that V_1 represents a defined variable in the script, which contains the data written by P_2. Therefore, the processing element P_1 which produces V_1 is the computing processing element having persistent output data. It is represented by hasOutput = true value. RHS of Fig. 10.C shows the processing chain with the changed values of relevant properties.

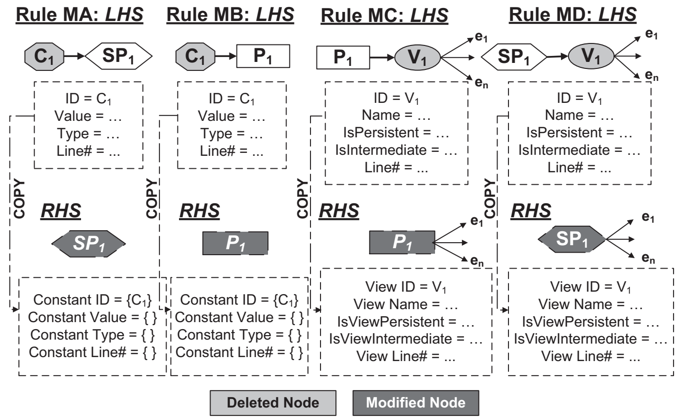


Fig. 11. Re-write rules for model modification.

E. Model Modification Re-Write Rules

In the provenance graph model described in Section III-A, both source and computing processing elements have a view as an output. Therefore, the initial workflow provenance graph based on this model can be further reduced by discarding the views. Moreover, the constants read by the processing elements can also be omitted. To ensure that no information is lost, we copy the properties of the view or constant node to the corresponding source or computing processing element node. Therefore, to apply these rules, we change our provenance graph model described in Section III-A. The new model has two types of nodes: 1) source and 2) computing processing element. Both source and computing processing elements include properties of constant and view nodes so that we can copy these values of constants and views to the corresponding properties in the corresponding processing element.

Fig. 11 shows all four model modification rules. Rule *MA* unifies a constant node with the following source processing element and deletes the constant node. If a match is found, the rule *MA* copies all properties of the constant node C_1 to the corresponding properties of the source processing node SP_1 and deletes the constant node C_1 eventually. Since several constant nodes might be connected with the same source processing element, source processing element maintains an array or a list for keeping the values of the constant nodes properties. The dotted line in SP_1 refers to the source processing element based on the new modified model. Rule *MB* unifies a constant node with the following computing processing element. The computing processing element also maintains an array of values of the constant nodes properties.

The other two rules, *MC* and *MD*, unify a view node with the preceding computing processing element and source processing element, respectively, and discard the view node. Rules *MC* and *MD* also ensure that the outgoing edges from the view node, i.e., $e_1, \ldots, e_n$, are now connecting from the respective processing elements.

After applying all these re-write rules, we have the workflow provenance graph. Fig. 12 shows the workflow provenance graph after being transformed from the graph shown in the bottom part in Fig. 6. The workflow provenance graph is significantly more compact than the initial one and it also transforms all control dependences into data dependences.

This article has been accepted for inclusion in a future issue of this journal. Content is final as presented, with the exception of pagination.

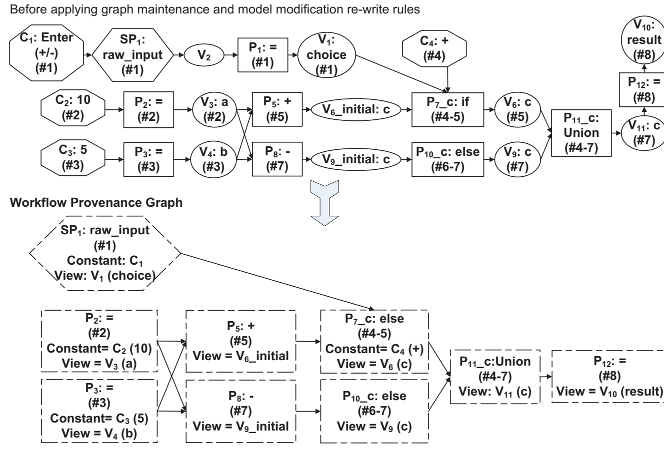


Fig. 12. Workflow provenance graph after applying re-write rules on the provenance graph shown in the bottom in Fig. 6.

F. Discussion

The concepts of workflow provenance inference as proposed in this paper are based on Python scripts. The Python program language is block-structured with user-defined functions and control-flow constructs, such as loops and conditional branching. Many programming languages share at least a subset of these ingredients, such as e.g., PHP or MATLAB scripts. While others have additional concepts like interfaces in Java or interaction with other users or components using messages instead of method calls in BPEL.

The core idea of the proposed approach is, however, independent of the used programming language. It is to automatically translate a control-flow coordinated program into a data-flow coordinated program. The mechanisms to perform this transition as proposed in this paper are limited to certain control flow coordination mechanisms, such as conditional branching, looping, and modularization. However, we have not addressed coordination mechanisms, including multiple parties as often done in BPEL or recursion besides others used as a means to realize an iteration.

Please be aware that provenance provides the origin of an individual result but does not explicate its semantics. The user has to interpret and understand the meaning of the processing steps and the used sources and constants. If semantic information, like e.g., metadata of the sources and their data structure, is available then this may help the user interpreting the provenance graph, but it is not part of the provenance graph as addressed in this paper.

V. Fine-Grained Provenance Inference

Based on the workflow provenance, either created and stored by a provenance-aware platform or inferred in a provenance-unaware platform as discussed in Section IV, the user can infer fine-grained data provenance. Fine-grained data provenance supports debugging during execution of the model as well as it can also be used to reproduce results.

A. Overview of the Concept

Fine-grained data provenance helps scientists to investigate the unexpected behavior of the model by keeping trace of output data products. Reproducible results validating the scientific model can also be achieved by facilitating fine-grained data provenance. Therefore, efficient management of fine-grained data provenance is in of utmost importance to the scientific community especially to the scientists handling massive and continuous data stream.

Fine-grained data provenance can be explicitly documented and stored in a database. However, in cases of massive streaming data, it requires storage space which becomes multiples of actual sensor data. To manage fine-grained data provenance in a storage-efficient manner, we proposed several methods to infer fine-grained provenance data [17], [22], [23]. These inference-based methods infer fine-grained provenance based on the given workflow provenance of the scientific model and timestamps attached to the data products.

In this section, we discuss the general principle of each of these methods and their applicability based on the model characteristics mentioned in Section I-D. To explain the general principle of each method, we introduce a few variables. A view V_i contains tuples $t_k{}^i$ where k indicates the point in time when it is entered into a database referred to as the transaction time. We define a window $w_j{}^i$ over the view V_i, which is an input view of processing element P_j. The window size is defined based on the number of data products, i.e., tuple-based window, or based on the interval in time units, i.e., time-based window. In either way, the window size of $w_j{}^i$ is referred to as $WS_j{}^i$. The processing element P_j is triggered after every TR_j time units known as trigger rate. The processing delay of P_j is referred to as δ_j.

B. Basic Provenance Inference

Basic provenance inference method [17] infers fine-grained data provenance in two phases: 1) backward computation and 2) forward computation. First, the scientist chooses an output data product for which the fine-grained provenance would be requested. Then, the backward computation phase is started. During the backward computation phase, this method takes the given workflow provenance graph into account. Facilitating the workflow provenance graph, i.e., window size over the input dataset, it reconstructs the original processing window over the input data products. Afterward, the second phase is executed. In the forward computation phase, the method establishes relationships between the input data products within the inferred window and the output data product based on the given workflow provenance, i.e., input–output ratio of the corresponding processing element.

Fig. 13 shows the different phases in the basic provenance inference method. Fig. 13(a) represents the data products in both input and output view V_1 and V_2, respectively. A data product arrives in V_1 after a fixed interval is elapsed, also known as sampling time, which is 2 time units in this example. The window size over V_1, $WS_1{}^1$ is 5 time units and the processing element P_1 triggers after every 5 time units, i.e., $TR_1 = 5$. We also assume that the processing delay of P_1, $\delta_1 = 0$. The user requests the provenance of t_{10} in the output view V_2.

Fig. 13(b) shows the reconstruction or backward computation phase. The transaction time of the chosen tuple is t_{10},

This article has been accepted for inclusion in a future issue of this journal. Content is final as presented, with the exception of pagination.

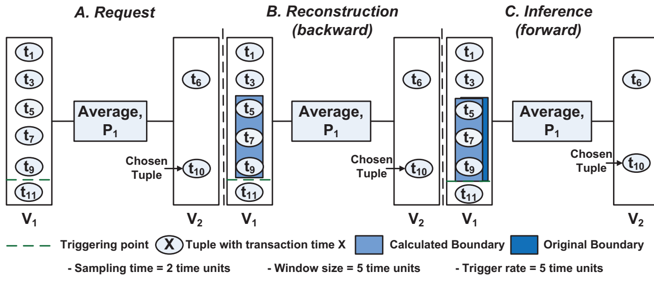


Fig. 13. Backward and forward computation in basic provenance inference technique.

which is the reference point to reconstruct the original processing window, also known as inferred window. The formula for calculating upper and lower bound of the inferred window is given below

$$\text{upperBound} = \text{referencepoint} - \delta_j \tag{B1}$$

$$\text{lowerBound} = \text{referencepoint} - \delta_j - {WS_j}^i \tag{B2}$$

where i and j be the index of the view and the processing element.

In the formula, the upper bound is always exclusive and the lower bound is inclusive. The given window size, ${WS_1}^1$ is 5 time units and processing delay, $\delta_1 = 0$ time unit. Therefore, based on (B1) and (B2), we retrieve the data products having transaction time within the boundary $[t_5, t_{10})$ from the view V_1. This set of data reconstructs the processing window, which is shown by the tuples surrounded by a light-shaded rectangle in Fig. 13(b).

The forward phase of the method establishes the relationship among the chosen output data product with the set of contributing input data products. This mapping is done by facilitating the input–output ratio of the processing element and the data products order in the respective view. P_1 takes all the input data products (i.e., n number of data products) and produces one output data making the input–output ratio $n : 1$. Therefore, we conclude that all the tuples in the reconstructed window contribute to produce the chosen tuple. In Fig. 13(c), the dark shaded rectangle shows the original processing window, which exactly coincides with the inferred processing window. Therefore, in this case, we achieve accurate provenance information. For processing elements with input–output ratio $1 : 1$, we have to identify the contributing input data product by facilitating the monotonicity in tuple ordering property in both views V_1 and V_2. This property ensures that input data products in view V_1 producing output data products in view V_2 in the same order of their transaction time and this order is also preserved in the output view V_2.

C. Probabilistic Provenance Inference

Probabilistic provenance inference [22] is a variant of the basic provenance inference method. The general principle remains the same. However, probabilistic inference can handle variation in the processing delay and data products arrival pattern by facilitating some prior knowledge about the delay and sampling time distributions.

Since the basic provenance inference method reconstructs the window based on the given processing delay δ_j for the jth processing element (B1) and (B2), the variation in the processing delay δ_j can result into inferring inaccurate provenance. As an example, in Fig. 13, if there is a processing delay, δ_1 of 1 time unit, the transaction time of the chosen tuple would be t_{11}. According to (B1) and (B2), the inferred window becomes $[t_6, t_{11})$. In this case, the inferred window excludes the data product with transaction time t_5, which actually contributed to produce the data product at t_{11}. Therefore, due to a variation in the processing delay, the basic provenance inference method provides inaccurate provenance. It is also possible to have a variation in the data products arrival pattern, i.e., sampling time. The basic provenance inference method is unable to cope with these variations.

Probabilistic provenance inference method overcomes this drawback by determining an optimal offset value, represented by O_j, for a given processing element P_j instead of using δ_j during backward computation phase. O_j refers to the distance in time between the reference point and the upper bound of the window. The optimal offset value avoids the problems of excluding contributing data products at the lower bound of the window and including noncontributing data products at the upper bound of the window in most cases based on the given model dynamics. The formula for calculating the inferred window is

$$\text{upperBound} = \text{referencepoint} - O_j \tag{C1}$$

$$\text{lowerBound} = \text{referencepoint} - O_j - {WS_j}^i \tag{C2}$$

where i and j be the index of the view and the processing element.

The calculation of O_j depends on the observed processing delay distribution δ_j and the sampling time distribution λ_i, where i is the index of the respective view participating in the processing element P_j. First, a probabilistic model is built, which represents data products arrival pattern with respect to the start of a processing window by facilitating Markov Chain modeling [33]. Since one of the characteristics of Markov Chain is to enter into an equilibrium state after repeated arrivals of data products, we can achieve a probability distribution of the distance in time between the point in time the window started and the arrival of the first tuple in that particular window. This distribution is known as α_i. We observe that if a particular value of α_i is less than the value of δ_j for the same processing window, the boundary of the window needs to be adjusted to achieve accurate provenance information. Since both δ_j and α_i are independent to each other, we can calculate their joint probability distribution and can compute the value of the optimal offset, O_j where the value of O_j maximizes the probability of $P(\alpha_i > \delta_j - O_j)$.

After computing the optimal offset O_j, the boundary of the inferred window can be calculated based on the formula given in (C1) and (C2). The forward computation phase is then executed to establish relationship between the data products within the inferred window and the output data product exactly in the same way as the basic provenance inference method does.

This article has been accepted for inclusion in a future issue of this journal. Content is final as presented, with the exception of pagination.

D. Multistep Probabilistic Provenance Inference

Multistep probabilistic provenance inference [23] is an extension of the probabilistic provenance inference method. While the working principle remains the same, the multistep method can infer fine-grained provenance data for an entire processing chain even in the presence of nonpersistent views.

In general, scientists facilitate a scientific workflow that is comprised of multiple processing elements to produce results. Some of these processing elements are intermediary steps and produce intermediate results, which might not be persistent in a database due to the lack of their reuse and sometimes ease of their calculation. Since the processing chain involves multiple intermediate steps and results of the intermediary steps are transient, the working mechanism of both backward and forward computational phase needs to be adjusted.

Like other inference-based methods, multistep provenance inference technique also depends on the given workflow provenance information. After receiving the provenance request for a chosen output data product, the backward computation phase is executed. During this phase, we observe the processing delay distributions δ of all processing elements, which allow us to calculate a window boundary on the materialized input view or source dataset. The formula to calculate the initial window boundary is given below

$$\text{upperBound} = \text{referencepoint} - \sum_{j=1}^{n}(\min(\delta_j)) \quad \text{(D1)}$$

$$\text{lowerBound} = \text{referencepoint} - \sum_{j=1}^{n}(\max(\delta_j)) - \sum_{j=1}^{n}(WS_j{}^i) \quad \text{(D2)}$$

where n be the total number of processing elements and i and j be the index of the view and the processing element.

Next, we execute the forward computation phase. In this phase, for each processing step, processing windows are reconstructed, i.e., inferred windows, and we compute the probability of existence of an intermediate output data product at a particular timestamp based on the δ distributions and other windowing constructs documented as workflow provenance. Multistep probabilistic method associates the output data product with the set of contributing input data products for each processing step with a probability. This process is continued till we reach the chosen data product for which provenance information is requested. The inferred provenance information has a cumulative probability P_C (< 1), which refers to the probability of being correctly inferred provenance. The detailed probability calculation is discussed in [23].

Furthermore, like the probabilistic provenance inference method, multistep probabilistic inference technique can also estimate the accuracy beforehand. To estimate the accuracy, we extend the approach discussed in the probabilistic inference method. In addition to compute α_i distribution, multistep probabilistic inference method is capable of computing λ_i distribution by facilitating a model that is built based on the same Markov Chain principle. The λ_i distribution is then used to compute the α_{i+1} distribution and this process continues till the end of the processing chain.

Test case parameters (in time units)					Inference-based Methods: Accuracy		
No.	Window size	Trigger	Avg. processing delay	Avg. sampling time	Basic	Probabilistic	Multi-step Probabilistic
1	10	10	1	3	57%	87%	86%
2	10	10	2	3	51%	75%	72%
3	10	10	1	4	63%	92%	90%

Fig. 14. Comparison in terms of accuracy between different inference-based methods.

Methods	Non-overlapping		Overlapping	
	Space consumed	Ratio	Space consumed	Ratio
Explicit	950	5.5	1925	11
Inference-based	175	1	175	1

Fig. 15. Comparison in terms of provenance storage consumption in KB between explicit and inference-based methods.

E. Performance of Inference-Based Methods

We evaluate three fine-grained provenance inference methods based on two factors: 1) accuracy and 2) storage cost. To compare the accuracy, fine-grained provenance is recorded explicitly, referred to as explicit method. This explicit provenance is used as the ground truth. We compare the accuracy between basic, probabilistic, and multistep probabilistic approach through a simulation.

The simulation is executed for 10 000 time units for several times with different parameters for a processing element, which performs average operation. Fig. 14 shows the test case parameters. To evaluate the multistep probabilistic method, we add another processing element in the chain performing the same operation with same set of parameters. We assume that both sampling time λ and processing delay δ distribution follow Poisson distribution. Fig. 14 reports the result for three different test cases. From this evaluation, our main findings are as follows.

1) Basic provenance inference method does not perform well under variable processing delay and sampling time. The accuracy is around 60%.
2) Probabilistic provenance inference performs reasonably well under variable model parameters. It achieves accuracy more than 85% if the average processing delay is significantly shorter than the average sampling time (see test cases 1 and 3). Otherwise, the accuracy gets lower (see test case 2 in Fig. 14).
3) Multistep probabilistic provenance inference achieves almost the same level of accuracy as the probabilistic method does.
4) The longer the processing delay, the higher the chance of getting low accuracy.
5) The longer the sampling time, the higher the chance of getting high accuracy.

Furthermore, we compare the storage requirement of the inference-based approaches with the explicit method of documenting provenance. A real dataset[16] reporting electrical conductivity of ground water, collected by the RECORD project is used for this purpose. The input dataset contains 3000

[16] Available at http://data.permasense.ch/topology.html#topology.

This article has been accepted for inclusion in a future issue of this journal. Content is final as presented, with the exception of pagination.

tuples consuming 720 KB. Our experiments are conducted for a single processing step performing an average operation with nonoverlapping windows, i.e., window size of 5 and trigger rate 5, and with overlapping windows, i.e., window size of 10 and trigger rate 5. The result is reported in Fig. 15. Basic and probabilistic provenance inference method have the same storage cost and they are referred to as inference-based methods. We have not applied the multistep probabilistic method on this dataset since there is only one processing step.

Fig. 15 shows the storage cost to maintain fine-grained provenance data for different methods. In case of nonoverlapping windows, the inference-based methods take almost 6 times less space than the explicit method. In case of overlapping windows, since the trigger is the same, it also produces as many output tuples as produced in the nonoverlapping case. The storage cost of inference-based methods only depends on the number of input and output data products. Therefore, the storage consumed in overlapping case by the inference-based methods remains the same. However, the consumed storage space for the explicit method gets bigger due to the larger window size and overlapping windows. Therefore, in the overlapping case, the inference-based methods take 11 times less space than the explicit method. This ratio of course will vary based on the window size, overlapping between windows, and number of output data products. In general, the bigger the window and overlapping between windows, the higher the ratio of space consumption between explicit and inference-based methods.

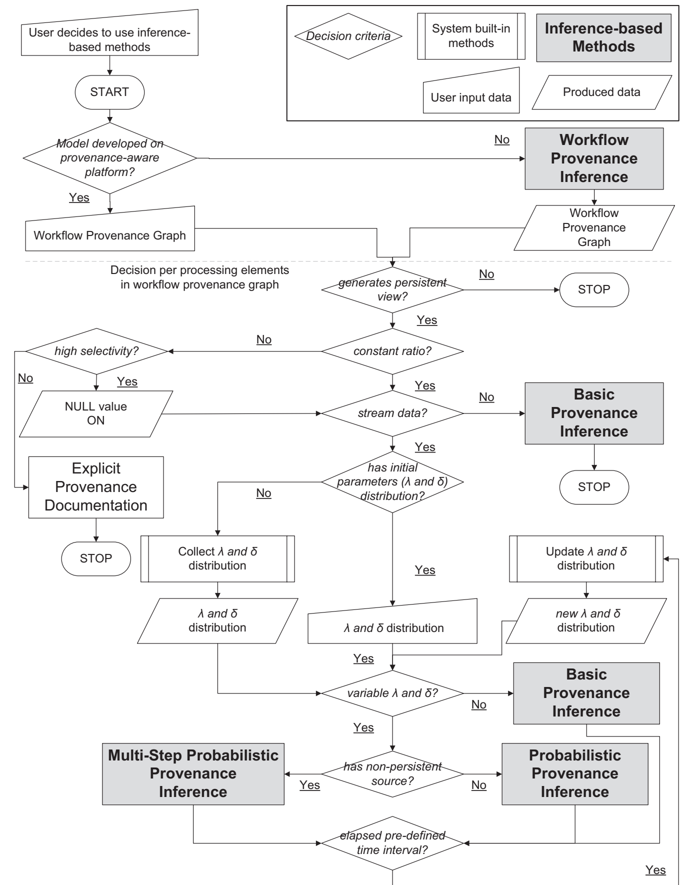


Fig. 16. Flow chart explaining the self-adaptability mechanism.

F. Self-Adaptability Mechanism

In Section I-D, we discussed the characteristics of different entities associated with a scientific model at both design and execution phase, which should be addressed by the proposed framework. During the design phase, depending on the model developing platform, the proposed framework needs to decide whether to apply workflow provenance inference method or not. After building the workflow provenance, the framework is capable of inferring fine-grained data provenance using three different inference-based methods described in Section V.

However, each of these fine-grained inference-based methods has their own pros and cons. The basic inference method has less complexity but achieves lower accuracy when the model parameters, such as processing delay and sampling time are variable as shown in Fig. 14. On the other hand, the probabilistic and multistep probabilistic inference method are more complex but achieve higher accuracy comparatively (see Fig. 14). Moreover, multistep probabilistic technique can infer provenance for a large processing chain with some nonpersistent intermediate views, which cannot be handled by the probabilistic method.

Therefore, the proposed framework would be more efficient if it could decide autonomously when to apply a particular inference-based method based on the model parameters. This ability is referred to as the self-adaptability. The self-adaptability of the framework not only used to choose the most suitable method but also to assess the variation in the model parameters once the model is running and can decide switching from one method to another if necessary.

Fig. 16 shows the flow chart, which explains the decision process executed by the self-adaptability mechanism. The process starts when a scientist decides to use the proposed framework. First, the mechanism considers the development platform of the model. If the model is developed using a provenance-aware platform, the workflow provenance graph is readily available. Otherwise, the self-adaptability mechanism decides to apply workflow provenance inference technique to build the workflow provenance graph. The documented characteristics of processing elements in the workflow provenance graph are used later in the decision process.

The next phase of the self-adaptability mechanism is executed per processing elements in the workflow provenance graph. First, it considers whether the processing element generates a persistent view or not. If the particular processing element does not produce a persistent view, the decision making process stops and considers the next processing element. Otherwise, it considers the input–output ratio of the given processing element in the next step. If the input–output ratio of the given processing element is variable like selection operations in a database, the decision tree then considers the selectivity rate, i.e., the percentage of input data products to be selected for processing within a processing window if the given condition is met. If the processing element has a high selectivity rate, then it switches NULL value mode ON, which

This article has been accepted for inclusion in a future issue of this journal. Content is final as presented, with the exception of pagination.

refers to the inclusion of null data products in the output view if the corresponding input data product is not selected for the processing. The inclusion of null products in the output view ensures that the output data product is created in the same order as the appearance of the contributing input data product and thus, the inference-based methods can be applied. If the processing element has a low selectivity rate, inclusion of null data products in the output view will incur more overhead and therefore, the self-adaptability mechanism decides to use explicit method for the given model.

After checking the selectivity rate, if the mechanism decides that inference-based methods can be applied on the given model (e.g., high selectivity rate) or if it finds that the given processing element has constant input–output ratio, it executes the next step. In this step, the decision process checks the type of available data, i.e., streaming or offline. If the model uses offline data for calculation, the decision process selects the basic inference method as the most suitable one. Otherwise, in cases of streaming data, it checks whether there exists distributions of two parameters, such as processing delay and sampling time as these are needed to apply the other inference-based methods. If these distributions do not exist, the model executing system collects this information for a pre-defined time interval during the actual execution time and prepares the required distribution information. After having the computed distributions at run-time or available distributions from the previous runs, the decision process checks the nature of these distributions. If it finds that both processing delay and sampling time are never changed, it chooses basic method to infer provenance. Otherwise, the self-adaptability mechanism checks whether the processing element has a nonpersistent view as the source or not. If yes, it means that the given processing element is the last one in the processing chain and there are some nonpersistent intermediate views. Therefore, in this case, the most suitable method to infer fine-grained data provenance is multistep probabilistic inference method. Otherwise, the decision process selects probabilistic method.

The self-adaptability mechanism always keeps track of the variation in the processing delay and sampling time distribution so that it can adjust its decision based on the recent executions of the system. To do that, it keeps updating the distributions and after a pre-defined time interval, it again executes the decision process by checking the nature of the updated distribution. In this way, the self-adaptability of the framework always ensures to apply the most suited method for the given scientific model based on the model dynamics.

VI. Use Case Evaluation

A. Quantitative Analysis

To evaluate the proposed provenance-aware framework, we introduced the use case based on the model estimating global water demand, described in Section II. The model used in the use case facilitates Python to implement different activities. Therefore, the model is built in a provenance-unaware platform. Moreover, these activities exhibit both control-flow and data-flow based coordination mechanism. Since the model is built in a provenance-unaware platform and possesses both control and data dependences between different activities, it is an appropriate choice to evaluate the proposed workflow provenance inference discussed in Section IV.

We build the initial provenance graph with 438 nodes based on the Python script having 116 lines of code used in the model. After applying the re-write rules explained in Section IV, the inferred workflow provenance graph consists of 139 nodes, which shows a significant reduction in the graph size by more than 300%.

Next, we infer fine-grained data provenance based on inferred workflow provenance of the model. To infer fine-grained data provenance, we need to import input data products in a database first. In this use case, there are more than 3000 PCRaster[17] maps containing input data products. We create a SQLite[18] database that contains tables for each persistent view found in the workflow provenance graph and then populate these tables with the values transformed from the map files. Further, we attach a timestamp to every value based on the data collection time. The size of the database for the use case is around 40 GB.

After importing the input data products, the model is executed and output data products are produced and stored into the same database. The input data products in this use case are collected and stored before the execution takes place. Therefore, these are offline data. Based on the self-adaptability mechanism introduced in Section V-F, we choose to apply basic provenance inference mechanism for this use case.

The user initiates the inference phase by choosing a particular value for which he wants to have fine-grained data provenance. Each value is characterized by its data collection time (year, month) and cell position in the (x, y) co-ordinates. Having this input from users, we apply the basic provenance inference method [17]. The accuracy of inferred fine-grained provenance inference is 100%, and the inference method does not consume any extra storage space to store provenance data.

B. Qualitative Analysis

We had several meetings with two scientists who developed this geoscientific model used in the use case. In the first meeting, we presented our approach of inferring provenance information and collected related data and scripts. Later, we developed our prototype and tested it with the given script.

After finalizing the prototype, we had another interview with the scientists to ask them several open-ended questions. We evaluate the proposed approach on the basis of three features: 1) extensibility; 2) debugging-friendliness; and 3) reproducibility.

1) Extensibility: Extensibility refers to the ability to handle different Python scripts and building workflow provenance graph out of them. Our prototype can handle varieties of Python scripts using different libraries. However, a user has to provide a few basic information on each method when it is called during the first run of the model only. These includes whether the function reads persistent data or not (e.g.,

[17] Available at http://pcraster.geo.uu.nl/.

[18] Available at http://www.sqlite.org/.

This article has been accepted for inclusion in a future issue of this journal. Content is final as presented, with the exception of pagination.

true/false) and whether the function writes persistent data or not (e.g., true/false) as well as the input–output ratio of the function.

Question: To what extent do you think that the extensibility of the proposed approach is helpful?

Feedback: The proposed approach is generic in the sense that it can handle varieties of Python scripts and builds workflow provenance graph out of those. However, at the very first run, the user has to enter method-specific information, which might be time-consuming and also requires some training for users.

2) Debugging-Friendliness: Debugging-friendliness refers to the suitability of a provenance graph for debugging purposes. Both workflow and fine-grained provenance graph can be used for debugging purposes. The workflow provenance graph shows the flow of the program, thus, can be used for code-level debugging. In addition, fine-grained provenance graph refers to the input data products and hence, can be used for value-level debugging.

Question: Have you ever experienced the need for a graph-based debugging tool? To what extent do you think that the provenance graphs are useful as a debugging tool?

Feedback: Usually, the scientists use the debugging tool which comes with the development environment. However, they appreciate the idea of debugging their code and the model using provenance graphs. Workflow provenance graph enables the code-level debugging, which is useful to determine the efficiency of the code, i.e., finding out code repetition. It is also useful to compare two different versions of the code expected to produce the same value. In addition, value-level debugging facilitating fine-grained provenance graph provides easy access to the actual data. It also proves beneficial when tracing back for identifying missing values in the file.

3) Reproducibility: Reproducibility in this paper means the ability to regenerate data items, i.e., for every processing element P, executed on an input dataset I at time t resulting in output dataset O; the re-execution of processing element P at any later point in time t' (with $t' > t$) on the same input dataset I will generate exactly the same output dataset O.

Question: To what extent do you think that fine-grained provenance graph is useful to achieve reproducibility? How do you use your reproducible results?

Feedback: Fine-grained provenance graph shows original data values contributed to produce the result, which helps to achieves reproducibility. In practice, reproducible results might be useful to explain the mechanism of the model to one of the other scientists from the same group.

C. Summary

Our quantitative evaluation shows that we can successfully infer both workflow and fine-grained data provenance. The provenance information is explicated as a graph. The set of re-write rules discussed in Section IV transforms the control dependences into data dependences and also makes the final graph more compact, which is appreciated by the scientists during the interview.

The scientists admit that the use of provenance graphs for debugging purposes makes the proposed framework more enticing to the scientific community. It is a common scenario that researchers waste a lot of time to wonder about a particular value. The fine-grained provenance graph comes very useful in these cases. Furthermore, workflow provenance graph can provide an overview on the complete model visually and can save a lot of time of researchers. Moreover, the researchers think that the proposed framework is extensible so that it can be used in any other use case. Fine-grained data provenance inference can achieve reproducibility, which can be used to validate the outcome of the model. In sum, the proposed inference-based framework is useful to the researchers in practice.

VII. Related Work

Data provenance has many applications in different domains. It can be used to validate scientific models. It can also be used as a replication recipe for the output data in a database system. Furthermore, provenance is seen as a type of data quality measure in geospatial domains. Therefore, provenance is a widely studied topic by the researchers from different domains.

Researchers have paid attention to make provenance-aware workflow engines. A provenance model described in [34] can collect provenance automatically during run time. This model is an extension of the Kepler [9] workflow engine. A layered model to represent workflow provenance is introduced in [35], which facilitates windows workflow foundation[19] as the workflow engine. These techniques are used in a provenance-aware platform but they do not offer any functionalities to infer workflow provenance without any human intervention. However, the proposed framework builds the workflow provenance graph automatically based on a given script.

One of the existing works in this direction is the usage of a program dependence graph (PDG) to get an overview of the model. A PDG makes explicit both the data and control dependences for each statement in a program [36]. A system dependence graph extends the definition of a PDG and it is capable of providing data and control dependences for multiprocedure programs [31]. To investigate a model with anomalies, scientists need to interpret control dependences in these graphs by themselves, which might be a tedious job due to the high complexity of the model. Our proposed framework can infer workflow provenance by transforming these control dependences into data dependences.

Provenance is also discussed in the context of geospatial domain. Yue *et al.* [37] proposed an approach to capture provenance data automatically using semantic web technologies. The provenance data have been stored in a resource description framework (RDF) triple store, and have been queried using SPARQL based on a geospatial data provenance ontology. Furthermore, a provenance framework has also been proposed in [38] for geoprocessing workflows. This framework provides provenance at different levels of a given geoscientific model. Yue *et al.* [39] reported an approach that enables interoperability for the collected provenance information in

[19] Available at http://www.windowsworkflowfoundation.eu/.

This article has been accepted for inclusion in a future issue of this journal. Content is final as presented, with the exception of pagination.

a service-oriented GIS architecture. Another method of capturing provenance has been discussed in [40]. In this paper, authors build provenance traces of the computation of snow-covered area by monitoring system-level calls from different running processes. Like these aforesaid methods, the proposed framework provides provenance at different levels facilitating static analysis of the script as well as fine-grained provenance inference methods. However, the proposed framework never stores any explicit provenance information.

Furthermore, provenance has also attracted researchers from the database systems. There are several existing methods which maintain fine-grained provenance data explicitly in a relational database. LIVE [19] is a complete DBMS, which preserves explicitly the lineage of derived data items in form of boolean algebra. In sensornet republishing, Park and Heidemann [20] used an annotation-based approach to represent data provenance explicitly, which is expensive in terms of storage. Furthermore, these approaches are not capable of self-adapting themselves based on the model characteristics.

There are a few research for collecting provenance in a provenance-unaware platform. Miles *et al.* [41] proposed a methodology, known as PrIMe, that can adapt applications to make them provenance-aware by exposing application information documented through a series of steps and by modifying the application design. Groth *et al.* [42] proposed a technique that can reconstruct provenance of the manipulations done over the data in a provenance-unaware system like excel sheet or a programming tool like R. This approach used a library of basic transformations to infer and reconstruct provenance for a particular value. Since it requires predefined possible transformations to reconstruct the data provenance, this approach is not easy to apply in different platforms. Silles and Runnalls [43] proposed a variant of R interpreter, CXXR, which can maintain and represent collected provenance information. Miles [44] proposed to document provenance by modifying the source code of a program automatically. It provides fine-grained data provenance after executing the script. However, one distinguishing factor is that our approach provides both workflow provenance and fine-grained data provenance.

Our proposed framework can capture workflow provenance automatically based on a given Python script. In StarFlow, Angelo *et al.* [18] proposed a method which can build provenance trace at functional level for a Python script. However, this tool cannot explicate the data dependences within a function. There are some other existing tools and packages,[20,21,22,23] which show the call graph based on a given Python script, i.e., dependency among different modules used in the script. However, neither of these tools can provide data dependences for a given script.

VIII. Conclusion

Scientists understand the importance of provenance data. However, provenance data were rarely maintained due to the lack of time and proper training to use the workflow engines and other tools. Furthermore, an integrated framework to provide both workflow and fine-grained data provenance was much needed due to the increasing popularity of data-intensive applications. Therefore, we proposed an inference-based framework to manage provenance data especially for geoscientific applications. We introduced an approach that can build workflow provenance graph automatically based on a given Python script. Since every scientific model has different characteristics, we incorporated the self-adaptability mechanism to the framework, which can select the appropriate method to infer fine-grained data provenance based on the model parameters. We evaluated the proposed framework using a hydrological model facilitating geospatial data. In future, we plan to improve user interface of the framework as well as to add new functionalities. Overall, our proposed framework helps scientists to use provenance information with minimal effort in time, training, and storage cost.

[20] Available at http://furius.ca/snakefood/.
[21] Available at http://pycallgraph.slowchop.com/.
[22] Available at http://www.tarind.com/depgraph.html.
[23] Available at http://pypi.python.org/pypi/Sumatra.

Acknowledgment

The authors would like to thank Y. Wada and L. P. H. van Beek from Utrecht University, Utrecht, The Netherlands, for providing the Python script used in [24] as well as related input data to showcase a complete demonstration.

References

[1] H. B. Newman, M. H. Ellisman, and J. A. Orcutt, “Data-intensive e-Science frontier research,” *Commun. ACM*, vol. 46, no. 11, pp. 68–77, Nov. 2003.
[2] P. Yue and L. He, “Geospatial data provenance in cyberinfrastructure,” in *Proc. IEEE 17th Int. Conf. Geoinformat.*, Aug. 2009, pp. 1–4.
[3] D. Lanter, “Design of a lineage-based meta-data base for GIS,” *Cartography Geograph. Inf. Sci.*, vol. 18, no. 4, pp. 255–261, 1991.
[4] P. Buneman, S. Khanna, and T. Wang-Chiew, “Why and where: A characterization of data provenance,” in *Proc. Int. Conf. Database Theory*, 2001, pp. 316–330.
[5] Y. L. Simmhan, B. Plale, and D. Gannon, “A survey of data provenance in e-Science,” *SIGMOD Rec.*, vol. 34, no. 3, pp. 31–36, 2005.
[6] P. Buneman and W. C. Tan, “Provenance in databases,” in *Proc. ACM SIGMOD Int. Conf. Manage. Data*, 2007, pp. 1171–1173.
[7] M. Shields, “Control- versus data-driven workflows,” in *Workflows for e-Science*. New York, USA: Springer-Verlag, 2007, pp. 167–173.
[8] W. Aalst and K. Hee, *Workflow Management—Models, Methods, and Systems*. Cambridge, MA, USA: MIT Press, 2002.
[9] B. Ludascher, I. Altintas, C. Berkley, D. Higgins, E. Jaeger, M. Jones, E. Lee, J. Tao, and Y. Zhao, “Scientific workflow management and the Kepler system,” *Concurrency Comput., Pract. Exper.*, vol. 18, no. 10, pp. 1039–1065, 2006.
[10] Y. L. Simmhan, B. Plale, and D. Gannon, “Karma2: Provenance management for data driven workflows,” *Int. J. Web Services Res.*, vol. 5, pp. 1–23, Apr. 2008.
[11] T. Oinn, M. Addis, J. Ferris, D. Marvin, M. Greenwood, T. Carver, M. Pocock, A. Wipat, and P. Li, “Taverna: A tool for the composition and enactment of bioinformatics workflows,” *Bioinformatics*, vol. 20, no. 17, pp. 3045–3054, Jun. 2004.
[12] S. Callahan, J. Freire, E. Santos, C. Scheidegger, C. Silva, and H. Vo, “Vistrails: Visualization meets data management,” in *Proc. ACM SIGMOD Int. Conf. Manage. Data*, 2006, pp. 745–747.
[13] B. Babcock, S. Babu, M. Datar, R. Motwani, and J. Widom, “Models and issues in data stream systems,” in *Proc. ACM Symp. Principles Database Syst.*, 2002, pp. 1–16.
[14] D. Abadi, D. Carney, U. Çetintemel, M. Cherniack, C. Convey, S. Lee, M. Stonebraker, N. Tatbul, and S. Zdonik, “Aurora: A new model and architecture for data stream management,” *VLDB J.*, vol. 12, no. 2, pp. 120–139, 2003.

This article has been accepted for inclusion in a future issue of this journal. Content is final as presented, with the exception of pagination.

[15] D. J. Abadi, Y. Ahmad, M. Balazinska, U. Cetintemel, M. Cherniack, J.-H. Hwang, W. Lindner, A. S. Maskey, A. Rasin, E. Ryvkina, N. Tatbul, Y. Xing, and S. Zdonik, "The design of the borealis stream processing engine," in *Proc. 2nd Biennial Conf. Innovative Data Syst. Res.*, 2005, pp. 277–289.

[16] N. A. Lynch, *Distributed Algorithms*. San Mateo, CA, USA: Morgan Kaufmann, 1996.

[17] M. R. Huq, A. Wombacher, and P. M. G. Apers, "Inferring fine-grained data provenance in stream data processing: Reduced storage cost, high accuracy," in *Proc. Int. Conf. Database Expert Syst. Appl.*, LNCS 6861. 2011, pp. 118–127.

[18] E. Angelino, D. Yamins, and M. Seltzer, "Starflow: A script-centric data analysis environment," in *Proc. Int. Provenance Annotat. Workshop*, 2010, pp. 236–250.

[19] A. Sarma, M. Theobald, and J. Widom, "LIVE: A lineage-supported versioned DBMS," in *Proc. Int. Conf. Sci. Stat. Database Manage.*, LNCS 6187. 2010, pp. 416–433.

[20] U. Park and J. Heidemann, "Provenance in sensornet republishing," in *Proc. 2nd Int. Provenance Annotat. Data Process.*, 2008, pp. 280–292.

[21] M. R. Huq, A. Wombacher, and P. M. G. Apers, "Facilitating fine grained data provenance using temporal data model," in *Proc. 7th Int. Workshop Data Manage. Sensor Netw.*, 2010, pp. 8–13.

[22] M. R. Huq, P. M. G. Apers, and A. Wombacher, "Probabilistic inference of fine-grained data provenance," in *Proc. Int. Conf. Database Expert Syst. Appl.*, LNCS 7446. 2012, pp. 296–310.

[23] M. R. Huq, P. M. G. Apers, and A. Wombacher, "Fine-grained provenance inference for a large processing chain with non-materialized intermediate views," in *Scientific and Statistical Database Management* (Lecture Notes in Computer Science), vol. 7338. New York, USA: Springer-Verlag, 2012, pp. 397–405.

[24] Y. Wada, L. P. H. van Beek, D. Viviroli, H. H. Dürr, R. Weingartner, and M. F. P. Bierkens, "Global monthly water stress: II. Water demand and severity of water," *Water Resour. Res.*, vol. 47, no. 7, p. W07518, 2011.

[25] F. Portmann, S. Siebert, C. Bauer, and P. Döll, "MIRCA2000—Global monthly irrigated and rainfed crop areas around the year 2000: A new high-resolution data set for agricultural and hydrological modelling," *Global Biogeochem. Cycles*, vol. 24, no. 1, p. GB1011, 2010.

[26] S. Siebert and P. Döll, "Quantifying blue and green virtual water contents in global crop production as well as potential production losses without irrigation," *J. Hydrol.*, vol. 384, nos. 3–4, pp. 198–217, 2010.

[27] J. Rohwer, D. Gerten, and W. Lucht, "Development of functional types of irrigation for improved global crop modelling," Potsdam Inst. Climate Impact Research, Potsdam, Germany, PIK Rep. 104, 2007.

[28] L. P. H. van Beek, Y. Wada, and M. F. P. Bierkens, "Global monthly water stress: I. water balance and water availability," *Water Resour. Res.*, vol. 47, no. 7, p. W07517, 2011.

[29] A. Wombacher, "Data workflow—a workflow model for continuous data processing," Centre for Telematics and Information Technology Univ. Twente, Enschede, The Netherlands, Tech. Rep. TR-CTIT-10-12, 2010.

[30] P. M. Luc Moreau. (2011 Oct. 18). *The PROV Data Model and Abstract Syntax Notation* [Online]. available: http://www.w3.org/TR/prov-dm/

[31] S. Horwitz and T. Reps, "The use of program dependence graphs in software engineering," in *Proc. Int. Conf. Softw. Eng.*, 1992, pp. 392–411.

[32] R. Cytron, J. Ferrante, B. Rosen, M. Wegman, and F. Zadeck, "An efficient method of computing static single assignment form," in *Proc. ACM SIGPLAN-SIGACT Symp. Principles Program. Lang.*, pp. 25–35, 1989.

[33] C. M. Bishop, *Patter Recognition and Machine Learning*. New York, USA: Springer-Verlag, 2006.

[34] S. Bowers, T. M. McPhillips, and B. Ludäscher, "Provenance in collection-oriented scientific workflows," *Concurrency Comput., Pract. Exper.*, vol. 20, no. 5, pp. 519–529, 2008.

[35] R. Barga and L. Digiampietri, "Automatic capture and efficient storage of e-Science experiment provenance," *Concurrency Comput., Pract. Exper.*, vol. 20, no. 5, pp. 419–429, 2008.

[36] J. Ferrante, K. Ottenstein, and J. Warren, "The program dependence graph and its use in optimization," *ACM Trans. Program. Lang. Syst.*, vol. 9, no. 3, pp. 319–349, 1987.

[37] P. Yue, J. Gong, and L. Di, "Augmenting geospatial data provenance through metadata tracking in geospatial service chaining," *Comput. Geosci.*, vol. 36, no. 3, pp. 270–281, 2010.

[38] P. Yue, Z. Sun, J. Gong, L. Di, and X. Lu, "A provenance framework for Web geoprocessing workflows," in *Proc. IEEE Int. Geosci. Remote Sens. Symp.*, Jul. 2011, pp. 3811–3814.

[39] P. Yue, Y. Wei, L. Di, L. He, J. Gong, and L. Zhang, "Sharing geospatial provenance in a service-oriented environment," *Comput., Environ. Urban Syst.*, vol. 35, no. 4, pp. 333–343, 2011.

[40] J. Dozier and J. Frew, "Computational provenance in hydrologic science: A snow mapping example," *Phil. Trans. R. Soc. A, Math., Phys. Eng. Sci.*, vol. 367, no. 1890, pp. 1021–1033, 2009.

[41] S. Miles, P. Groth, S. Munroe, and L. Moreau, "PrIMe: A methodology for developing provenance-aware applications," *ACM Trans. Softw. Eng. Methodol.*, vol. 20, no. 3, p. 8, 2011.

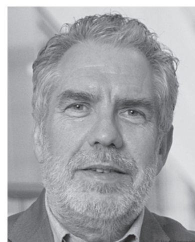
[42] P. Groth, Y. Gil, and S. Magliacane, "Automatic metadata annotation through reconstructing provenance," in *Proc. CEUR Workshop Semant. Web Provenance Manage.*, vol. 856. 2012, pp. 1–8.

[43] C. Silles and A. Runnalls, "Provenance-awareness in R," in *Proc. Int. Provenance Annotat. Workshop*, LNCS 6378. 2010, pp. 64–72.

[44] S. Miles, "Automatically adapting source code to document provenance," in *Proc. Int. Provenance Annotat. Workshop*, LNCS 6378. 2010, pp. 102–110.

Mohammad Rezwanul Huq received the Bachelor degree in computer science from the Islamic University of Technology, Dhaka, Bangladesh, in 2004, and the M.Sc. degree in computer engineering from Kyung Hee University, Seoul, Korea, in 2008. Since 2009, he has been pursuing the Ph.D. degree with the Database Group, University of Twente, Twente, The Netherlands.

His current research interests include managing data provenance in scientific applications. He is supervised by Prof. P. Apers and Dr. A. Wombacher.



Peter M. G. Apers has been a Full Professor in computer science with the University of Twente, Twente, The Netherlands, since 1985. He is involved in a large number of externally funded projects. Since 2002, he has been the Scientific Director of the Centre for Telematics and Information Technology (CTIT), University of Twente. With more than 400 researchers, he has focused on ICT as a technology and on ICT applications. He has taken the initiative to found NIRICT (Netherlands Institute for Research on ICT), the ICT institute of the three Dutch Universities of Technologies, of which he is Scientific Director from the start. He is a Scientific Leader of the National ICT Roadmap. He has been a member of the Executive Steering Board of EIT ICT Labs since 2010 and responsible for the co-location in the Netherlands.

Andreas Wombacher (M'02) received the M.Sc. and Ph.D. degrees in computer science from the Technical University of Darmstadt, Darmstadt, Germany, in 1996 and 2005, repectively.

He held a Post-Doctoral position with the University of Twente, Twente, The Netherlands, from 2004 to 2006, and a second Post-Doctoral position from 2006 to 2007 with the Cole Polytechnique Fedrale de Lausanne, Lausanne, Switzerland. He joined the Database Group, University of Twente, as an Assistant Professor, in 2007, working on workflow aspects of data processing.