ProvenanceCurious: A Tool to Infer Data Provenance from Scripts

Mohammad Rezwanul Huq, Peter M.G. Apers, Andreas Wombacher
Dept. of Computer Science, University of Twente
7522NB, Enschede, The Netherlands.
{m.r.huq, p.m.g.apers, a.wombacher} @utwente.nl

ABSTRACT

The increasing data volume and highly complex models used in different domains make it difficult to debug models in cases of anomalies. Data provenance provides scientists sufficient information to investigate their models. In this paper, we propose a tool which can infer fine-grained data provenance based on a given script. The tool is demonstrated using a hydrological model. The tool is also tested successfully handling other scripts in different contexts.

Categories and Subject Descriptors

E [**Data**]: Miscellaneous—*Data Provenance*

Keywords

Data provenance, Code analysis, Scientific models

1. INTRODUCTION

Data intensive applications are used to study and better understand complex systems such as physical, geographical, environmental, biological etc. [6]. In these applications, the data collection contains both in-situ data and streaming data. Scientists use this data fitting into their model describing processes in the physical world. During the execution of the model, imprecise or unexpected values could be generated occasionally due to the anomalies either in the data or in the model. To resolve this issue, scientists need to debug their scripts used for actual processing as well as to trace back values of the input data sources. Maintaining data provenance can help them in such a situation.

Data provenance refers to the derivation history of data starting from its input sources [1]. Fine-grained data provenance is defined at the value-level which refers to the determination of how a particular value has been created and processed starting from it's input values. It helps scientists to trace back values of the input data sources. Coarse-grained or workflow provenance is defined at the more higher level of granularity. Workflow provenance only captures association among different operations within the model allowing scientists to debug through their model. Both fine-grained and workflow provenance are necessary to identify the origin of the unexpected behavior of the model.

Maintaining data provenance explicitly requires a lot of effort from the users which is not desirable and appreciated. Therefore, inference of data provenance based on a given script has a high demand in the scientific community. A script consists of both control-flow and data-flow based operations. Execution of a control-flow based operation depends on the successful completion of previous operations [7] and it exhibits control dependences between operations. Handling the control-flow based operations (e.g. loop, conditional branching etc.) offers the major challenge to infer data provenance. Since data provenance only explicates the data dependences among the operations and the values, these control dependences between operations must be translated into the data dependences where the availability of actual values fires the next operation. In this paper, we describe the development of such a tool, known as *ProvenanceCurious*, which can overcome this challenge and can infer data provenance based on a given script.

The proposed tool builds the workflow provenance from the given script. Based on the workflow provenance and the availability of actual values from different data sources, it infers fine-grained data provenance. The inference of fine-grained data provenance allows the users to debug the model by specifying the value of interest in space and time. 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 scripts. Python[1] is widely-used to handle spatial and temporal data in the scientific community as well as in commercial products such as ArcGIS[2] which has inspired us to make this choice.

The proposed tool is demonstrated using a hydrological model estimating the global water demand [8]. The tool is also tested with other multi-procedure Python scripts in different contexts. Using the proposed tool, scientists are able to visualize workflow provenance, i.e. data dependences among operations. Moreover, they can also select individual values which are generated by executing the script and infer fine-grained provenance, i.e. input values those have been used to calculate the selected value. Especially in cases of outliers or missing values, it is a beneficial tool, providing the scientists sufficient information to investigate the unexpected behavior of the model.

[1]http://www.Python.org/
[2]http://www.esri.com/software/arcgis

2. PROVENANCE GRAPH MODEL

The proposed tool, *ProvenanceCurious*, infers data provenance and represents it as a graph known as *provenance graph*. A provenance graph G_p is a set of (V, E) where V denotes the set of nodes and E denotes the set of directed edges. A node represents either data or an operation. A directed edge connecting two nodes represents the data dependence of the target node on the source node. The graph model consists of different types of nodes. These are:

- **Constant**: represents any constant value taking part in an operation.
- **Source Processing Element**: represents any operation that either assigns a constant or reads data from the disk.
- **Computing Processing Element**: represents any operation that either computes a value based on its parameters or writes data into the disk.
- **View**: represents either any variable defined in the script or intermediate result generated by a process.

Every source and computing processing element generates a view. Further, a view or constant node can be used as an input for multiple processing elements. Each type of nodes has different properties. Some of the properties are important to explicate fine-grained provenance information. As for example, each processing element node has properties such as: i) *windows* which is defined over it's input views specifying the number of data products to be considered to process, ii) *trigger* which specifies how often the processing element is executed and iii) *hasOutput* which holds a boolean value indicating that the processing element produces persistent data (true) or not (false). Moreover, each view has some distinct boolean properties such as: i) *IsIntermediate* which is true if the view contains intermediate result set and false otherwise and ii)*IsPersistent* which is true if the view contains data products that are read from or written into the disk. The proposed tool infers the value of these properties while building the workflow provenance graph and facilitates this information to infer fine-grained data provenance.

3. SYSTEM DESCRIPTION

The proposed tool is comprised of five components: graphical interface, graph building engine, inference engine, database engine and customization engine. Fig. 1 shows these components, the inner modules of each component and the user interaction with the tool. Each component is represented by a rectangle with thick boundary. The other rectangles within a component represent a module belonging to that component. The ellipses represent outputs produced by the modules. However, the user can interact with only the shaded-ellipses. The dotted lines show the user input to a module specifying additional parameters required by the system.

3.1 Graphical Interface

The user interacts with the tool through the graphical interface. First, the user gives the input, i.e. a Python script, to the *import* module which then invokes the graph building engine to build the workflow provenance graph. The user also provides a few operation-specific information required by the graph building engine through the *operations info* module. Upon the completion of generating the workflow provenance graph, the user can also request to infer

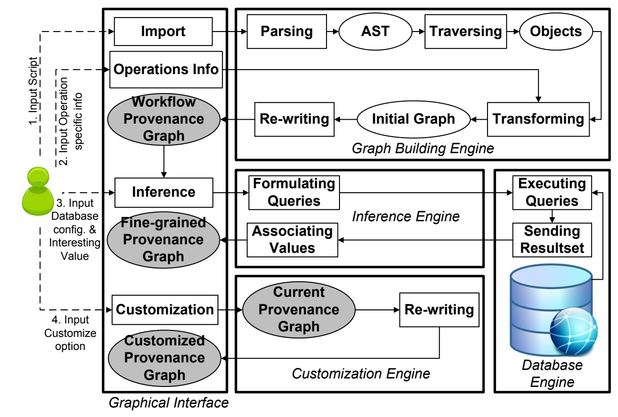


Figure 1: Different components of the system

the provenance for an interesting value in time and space. At this time, the user has to provide the actual location of the database that holds the values through the *inference* module. Furthermore, selecting a provenance graph either workflow or fine-grained, the user can also request to have a customized version of the selected graph by specifying an appropriate option via the *customization* module.

3.2 Graph Building Engine

The graph building engine generates the workflow provenance graph based on the given Python script. First, it parses the script based on a grammar. After parsing the script, it returns an abstract syntax tree (AST) for the given script. Then, it traverses 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 is created. At this time, it asks users to enter a few operation-specific information. The operation-specific information includes whether the operation reads persistent data or not (e.g. true/false) and whether the operation writes persistent data or not (e.g. true/false). Having the user input, it builds the initial workflow provenance graph based on the graph model discussed in Sec.2. Since the initial provenance graph captures all syntactical details of the script, the size of this graph becomes significantly large. Therefore, we apply a set of graph re-write rules on the initial graph to reduce the number of nodes and edges to achieve the final *workflow provenance graph*. The workflow provenance graph is then returned to the interface so that the user can interact with it.

3.3 Inference Engine

The inference engine is executed when the user requests to debug the model based on a given value. The inference engine also receives the required parameters such as the workflow provenance graph and database location from the user. The value requested by the user consists of several parameters depending on the underlying model. Suppose, for the model that estimates the global water demand [8], the value is characterized by it's timestamp (e.g. *yyyy-mm*) and cell position in 2-d co-ordinates. Having this input from users, the inference engine applies appropriate provenance inference algorithm [5], [4] to formulate the provenance queries.

The model reported in [8] contains only in-situ and static data. Therefore, in this case, we apply the basic algorithm described in [5]. After formulating the queries, it invokes the database engine to execute the queries on the actual values. The results are then returned to the inference engine. The inference engine associates the results containing actual values to the appropriate nodes of the workflow provenance graph to build the fine-grained provenance graph. Then, the fine-grained provenance graph is returned to the user.

3.4 Database Engine

The database engine executes the queries sent by the inference engine. At the time of execution, the *executing queries* module makes a connection to the underlying database, fetches relevant values and executes queries. After the execution, the results are sent to the *sending* module which returns the results to the inference engine for further action.

3.5 Customization Engine

The customization of a provenance graph can satisfy users with different level of understanding and objectives. The customization engine provides a handful options to customize a provenance graph. The user has to specify one of these options and based on the selected option the engine then applies appropriate rules to transform the current provenance graph into the customized version of it. One of the customization techniques is to group processes, known as *grouping process* customization. In this technique, if several processing nodes with intermediate results are sequentially participating to produce a persistent view of a computing process, then those intermediary processing nodes are grouped together with the process generating the persistent view.

3.6 Implementation

The proposed tool is implemented as an eclipse[3] plug-in and written in Java using the latest JDK version 1.7.0_05. The graphical interface is built by facilitating the eclipse modeling framework (EMF) and graphical modeling framework (GMF). The Python grammar and tree grammar used in modules *parsing* and *traversing* of the graph building engine are developed using the ANTLR tool[4]. Further, the ANTLR APIs for Java are used to build the abstract syntax tree (AST) and to create objects by traversing the AST. To build the provenance graph, we use the attributed graph grammar (AGG) engine[5] and it's APIs for Java. The set of re-write rules used in both graph building and customization engine are realized by facilitating AGG. Since our demonstration scenario contains actual values in a SQLite database[6], the database engine currently includes a JDBC driver for SQLite database engine. However, drivers for other databases can be easily plugged in.

4. DEMONSTRATION

4.1 Demonstration Scenario

To evaluate the proposed tool, a scientific data processing model in hydrological domain reported in [23] is introduced.

[3]http://www.eclipse.org/
[4]http://www.antlr.org/
[5]http://user.cs.tu-berlin.de/~gragra/agg/
[6]http://www.sqlite.org/

Freshwater is one of the most important resources for various human activities and food production. 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. We consider the model estimating global water demand reported in [8] as our demonstration scenario. The model calculates water demand globally for every year starting from 1960 to 2000 at a monthly resolution.

The Python script used in the model has 120 lines of code and calculates water demand based on more than 3000 input files. These input files are raster maps with 360×720 cells where each cell contains a value. The calculated water demands are also stored in similar types of raster maps. The values of these maps are stored into a SQLite database that consumes around 40 GB of memory. Executing the proposed tool for the complete script generates a workflow provenance graph consisting of 139 nodes.

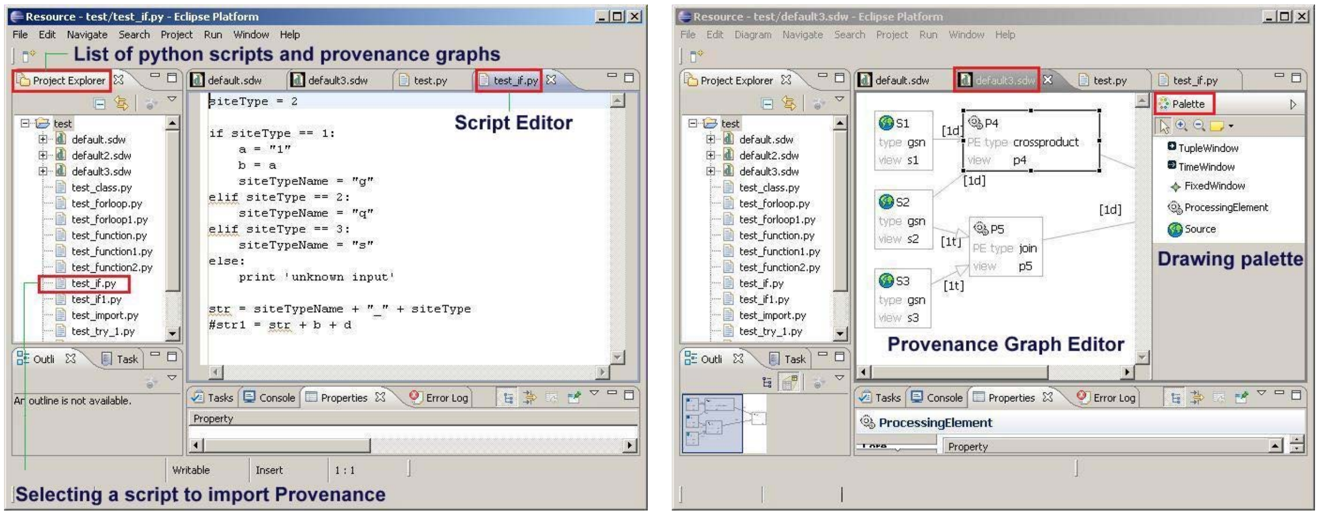
4.2 Execution of the Tool

The proposed tool, *ProvenanceCurious*, is executed as an eclipse plug-in application. Fig. 2(a) shows the initial window of the tool after the execution. In the left side of the window, the user could navigate through the project which contains a bunch of Python scripts and generated provenance graphs. The user can also update any Python script in the script editor. In Fig. 2(a), the script *test_if.py* is opened for editing. Later, the user can initiate the inference of data provenance by right-clicking on a particular script and choose *import provenance* option from the drop-down menu.

After initiating the inference of data provenance, the corresponding components and modules of the system are invoked (see Sec.3) to generate the workflow provenance graph. Fig. 2(b) shows a sample of the generated workflow provenance graph. The user can also edit few properties of the nodes and edges of the graph in the provenance graph editor. Moreover, it is also possible to add new nodes or edges from the drawing palette, positioned at the right side of the window shown in Fig. 2(b). Later, the user can request to infer fine-grained provenance graph for a particular cell in any of the output maps generated by the model. Based on this request, the tool infers all contributing input values to produce the value of the selected cell. Further, the graphical interface also allows customization of the current provenance graph in the editor based on the user-given option.

4.3 Discussion

The proposed tool translates the control dependences originating from a control-flow statement (e.g. loop) into data dependences to infer data provenance. In the demonstration scenario, we handle loops which read data, i.e. input maps, to produce the output map. However, loops updating data could be also translated into data dependences by inferring relevant properties such as windows and trigger of the processing elements. Furthermore, the proposed tool can handle variety of Python scripts including multi-procedure scripts and scripts with other programming constructs such as class definition, conditional branching etc. Since the system is built following the layered-design scheme, it is also possible to showcase the demonstration using scenarios without data repository. In that case, the inference engine and the database engine of the tool will be isolated from the system.



```
siteType = 2

if siteType == 1:
    a = "1"
    b = a
    siteTypeName = "g"
elif siteType == 2:
    siteTypeName = "q"
elif siteType == 3:
    siteTypeName = "s"
else:
    print 'unknown input'

str = siteTypeName + "_" + siteType
#str1 = str + b + d
```

(a) Script editor

(b) Provenance graph editor

Figure 2: The graphical interface of *ProvenanceCurious*

5. RELATED WORK

One of the existing work in this direction is the usage of a program dependence graph (PDG) to debug the model. A program dependence graph (PDG) makes explicit both the data and control dependences for each statement in a program [2]. A system dependence graph (SDG) extends the definition of program dependence graph and it is capable of providing data and control dependences for multi-procedure programs [3]. 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. Since the proposed tool translates control dependences into data dependences, scientists have a homogeneous graph with only the data dependences of the model which is easy to understand.

Furthermore, because of streaming data, the data volume always increases and it requires additional efforts in debugging models since a single output value could be influenced by a multitude of input values. The classical dependence graphs, i.e. SDG and PDG, cannot capture dependences among model input, model output values and associated operations. Existing debugging feature of the development environments and code analysis tools such as CodeSurfer[7], Frama-C[8] are capable of explicating this value-level dependency only at the run-time. Depending on the availability of input and output data from prior executions, the proposed tool allows scientists to visualize the dependences between the input and output values even without executing the model once again.

There are few existing tools[9,10] which show the call graph based on a given script, i.e. dependency among different modules used in the script. However, neither of these tools can transform the control dependences into data dependences and can infer fine-grained data provenance.

[7]http://www.grammatech.com/products/codesurfer/
[8]http://frama-c.com/
[9]http://furius.ca/snakefood/
[10]http://pycallgraph.slowchop.com/

6. CONCLUSION AND FUTURE WORK

In this paper, we demonstrate a tool that infers data provenance with minimal user annotation based on a given Python script and represents it as a provenance graph. The provenance graph provides an overview on data dependences and are useful for debugging scientific models. The approach is generally applicable to any procedural languages. In future, we plan to improve the interface of the proposed tool.

7. REFERENCES

[1] P. Buneman and W. C. Tan. Provenance in databases, in *ACM SIGMOD*, pages 1171–1173, ACM, 2007.

[2] J. Ferrante, K. Ottenstein, and J. Warren. The program dependence graph and its use in optimization, in *ACM Transactions on Programming Languages and Systems*, volume 9(3), pages 319–349, 1987.

[3] S. Horwitz and T. Reps. The use of program dependence graphs in software engineering, in *Int'l conference on Software engineering*, pages 392–411. ACM, 1992.

[4] 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*, LNCS, volume 7338, pages 397–405, 2012.

[5] 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 *Database and Expert Systems Applications*, LNCS, volume 6861, pages 118–127, 2011.

[6] H. B. Newman, M. H. Ellisman, and J. A. Orcutt. Data-intensive e-science frontier research, in *Communication of the ACM*, volume 46(11), pages 68–77, 2003.

[7] M. Shields. Control- versus data-driven workflows, in *Workflows for e-Science*, pages 167–173, 2007.

[8] Y. Wada et. al. Global monthly water stress: II. water demand and severity of water, in *Water Resources Research*, volume 47, 2011.