Revealing the Detailed Lineage of Script Outputs using Hybrid Provenance

Qian Zhang[1], Yang Cao[1], Qiwen Wang[2], Duc Vu[3], Priyaa Thavasimani[4], Timothy McPhillips[1], Paolo Missier[4], Peter Slaughter[5], Christopher Jones[5], Matthew B. Jones[5], Bertram Ludäscher[1]

[1]School of Information Sciences, University of Illinois at Urbana-Champaign, USA
[2]Department of Computer Science, University of Illinois at Urbana-Champaign, USA
[3]Department of Electrical and Computer Engineering, University of Illinois at Chicago, USA
[4]School of Computing Science, Newcastle University, UK
[5]National Center for Ecological Analysis and Synthesis, UCSB, Santa Barbara, USA

Abstract

We illustrate how combining retrospective and prospective provenance can yield scientifically meaningful *hybrid provenance* representations of the computational histories of data produced during a script run. We use scripts from multiple disciplines (astrophysics, climate science, biodiversity data curation, and social network analysis), implemented in Python, R, and MATLAB, to highlight the usefulness of diverse forms of retrospective provenance when coupled with prospective provenance. Users provide prospective provenance (i.e., the conceptual workflows latent in scripts) via simple YesWorkflow annotations, embedded as script comments. Runtime observables, hidden in filenames or folder structures, recorded in log-files, or automatically captured using tools such as noWorkflow or the DataONE RunManagers can be linked to prospective provenance via relational views and queries. The YesWorkflow toolkit, example scripts, and demonstration code are available via an open source repository.

1. Introduction

Scripts are widely used to automate scientific workflows. Provenance support for script-based workflows is expected to promote interpretation, openness, and reproducibility of compute- and data-intensive studies [24-27]. However, a gap often exists between the provenance that comes with a published dataset and the promised science-level explanations [24]. Part of the problem is that a retrospective provenance record (e.g., the events observed during the execution of a script on a particular input dataset) is only one of the elements needed to elucidate the process occurring during a computational experiment. Equally important is the general specification of the overall workflow. Following the nomenclature proposed in [23] and adopted in [1], we refer to the latter as *prospective provenance*, to distinguish it from (but also relate it to) what is commonly called *retrospective provenance*. Retrospective provenance observables include, e.g., the location, name, and contents of files read or written by a script, data created or updated during execution, parameter settings, and system-level information such as OS and software versions, timestamps, etc. Prospective provenance, on the other hand, captures the general "recipe" (i.e., workflow) used in a study. In workflow systems (e.g., Kepler, Taverna, Pegasus, Restflow, or Vistrails [19,16,17,21,9]) this workflow description is readily available as a directed graph, linking computational steps via dataflow channels, thus providing the desired high-level overview of a computational workflow. For script-based workflows, one could consider the script itself a form of prospective provenance, albeit one in which the underlying conceptual workflow and dataflow structure is usually obscured by details of the implementation. The motivation for developing YesWorkflow (YW) has been to allow users easily to expose high-level workflow models latent in scripts using simple user annotations embedded as script

comments [1]. We show here that the workflow models declared by users and extracted via the YW toolkit can be augmented and enriched by retrospective provenance observables, yielding various forms of *hybrid provenance* [2,3], i.e., fragments of workflow execution traces with structures provided by the user-declared YW models (prospective provenance) and with execution details filled in from one or more sources of runtime observables (retrospective provenance), cf. Figure 1.

Building on prior work [1-8], we demonstrate how a number of such provenance representations and products can be derived from a combination of prospective and retrospective provenance sources. We argue that the resulting hybrid views, queries, and visualizations are useful not only for documenting and explaining script-based scientific workflows to others, but also for enabling researchers to query and explore the derivation history of the data products yielded by script runs while computational studies are still underway. The techniques and examples discussed below are available as executable demonstrations via an open source repository [10].

2. Prospective and Retrospective Provenance Elements

We first describe the tools we use to capture prospective and retrospective provenance, respectively, followed by the approaches to combine them for hybrid query capabilities.

2.1 Modeling Scripts as Workflows with YW Annotations

YesWorkflow (YW) [1,2] aims to provide researchers developing script-based workflows with many of the benefits of scientific workflow systems. Script authors can embed YW annotations (`@BEGIN`, `@END`, `@IN`, and `@OUT`) within the comments of their scripts, thereby declaring relevant computational steps and the dataflow between them. The resulting workflow model provides a high-level prospective provenance graph that can be visualized and queried to reveal dataflow dependencies (Figure 1, left). Since prospective provenance is created via user-provided YW annotations, the appropriate level of modeling detail is controlled by users themselves.

2.2 Reconstructing Provenance with YW-Recon

The YesWorkflow model of a script can be augmented with runtime observations to yield hybrid provenance information. Some retrospective provenance from script runs can easily be reconstructed using resource *`URI` template* declarations: Provided that users name and organize their files, folders, and other resources systematically, YW can use declared URI templates to discover the actual files that were read or written at runtime [2]. For example, a URI template such as

```
@OUT Image @URI file:run/raw/{cassetteID}/{sampleID}/e{energy}/image_{no}.raw
```

declares that raw image files are found, relative to the folder in which the script was executed, within subfolders of `run/raw/`, and are organized first by cassette ID, then by sample ID, beam energy, and finally by image number. After script execution, by finding and matching the file paths to actual resources using the URI templates, YW can link the conceptual workflow entities—here the templates declaring the layout of input and output resources on the file systems—with observations made about the script run, in this case the actual folders and files created during the run.

In addition (or as an alternative) to the use of URI templates, a script author can employ user-defined *log-files* to capture runtime provenance observables at any level of granularity that is required to support the desired provenance queries. Using log files easily written by a script, YW can then harvest fine-grained retrospective information, e.g., at the record-level within a file, or the field-level within a record. Because the

structures of log file entries are declared via *@LOG template* declarations, the content of log entries can be chosen to maximize human readability of logs, while at the same time providing YW with machine-readable parsed information. Similar to URI templates, log templates are associated with other (here: @OUT) YW annotations, cf. Figure 6(a). A output resource name can be associated with multiple log entry templates. Each log entry template may include one or more template variables that distinguish multiple files (or other data sources) written by a script in a particular code block. As with URI template variables, log template variables are enclosed in curly braces. Following the run of a script, YW will use these log templates to discover the actual resources written during the run and harvest the information from the log entries, cf. Figure 6(b). In this way, YW supports reports of the results of running a script at various levels of granularity, e.g., at the script-, data-, file-, record-, field-, or function-level (provided prospective provenance declarations have been provided for each of those levels).

2.3 Recording Provenance with DataONE RunManagers (RM)

DataONE provides the RunManager provenance recording and management system for MATLAB and R script executions. A Run Manager overloads and thus intercepts file I/O operations to automatically capture runtime observables at the file level. The RunManager API provides functions[1] for capturing, searching, archiving, and sharing provenance. DataONE provides two RunManager implementations: the recordr R package[2] and the MATLAB toolbox[3]. The provenance captured during a script execution includes information about the script that was run, the files that were read or written, and details about the execution environment at the time of execution. A data package including the script itself, input files, and generated files associated with the run can easily be published to a repository within the DataONE network.

2.4 Code-level Provenance Capture with noWorkflow

The most detailed retrospective provenance recorder employed in these examples is noWorkflow [5]. The noWorkflow system (NW) employs a Python profiling library to capture fine-grained runtime provenance at the level of Python function call chains, variable assignments, and variable dependencies. The previously reported "provenance bridge" [3] between YW and NW provenance information is essential to some of the provenance use cases and demonstrations below [10].

3. Hybrid Provenance

Figure 1 provides an overview of our approach: A user models a workflow *W* using YW-annotations embedded as script comments. In this way, the user's conceptual workflow model can be bundled in a language-independent way with the executable program code[4]. The resulting YW model of a script constitutes a conceptual-level form of prospective provenance. The YW tool translates these annotations into an internal database format that can be exported as a ProvONE-compliant model, or in graph form, that can be rendered using Graphviz. The graphical rendering provides users with a high-level overview of *W*'s structure, including its underlying dataflow dependencies. Queries against the YW model (implemented, e.g., in Prolog or Datalog) can then be used to retrieve the upstream lineage of any final or intermediate data product *y*, (i.e.,

[1] github.com/DataONEorg/sem-prov-design/blob/master/docs/PROV-capture/Run-manager-API.rst
[2] DataONE recordr R package: https://github.com/NCEAS/recordr
[3] DataONE MATLAB toolbox: https://github.com/DataONEorg/matlab-dataone
[4] YW models can also be created in separate files, e.g., during workflow design, when an executable version of the script may not yet exist.

the subgraph of data and steps upstream of *y* in *W*), and to compute the downstream influence of *x* (i.e., the subgraph of the data derived from *x* along with the derivation steps. The YW toolkit can exploit different sources of retrospective provenance information (Fig. 1, right), consisting of runtime-observable events, to produce hybrid provenance (Figure 1, middle). This requires the definition of suitable "bridge rules" to establish associations between corresponding elements in the two types of provenance. We can distinguish different kinds of runtime observables, e.g., *file-level observables* from the DataONE RunManagers for MATLAB and R that overload file-IO commands, and *code-level observables*, e.g., from the noWorkflow system [5], which captures fine-grained runtime information at the level of function calls and changes of program variables while executing a Python script. Both the file-level and code-level runtime observables can be exploited for powerful hybrid queries. The former is the focus of our IDCC demonstration, while the latter has been demonstrated elsewhere [3]. Below we discuss some details of different "provenance bridges" we have developed.

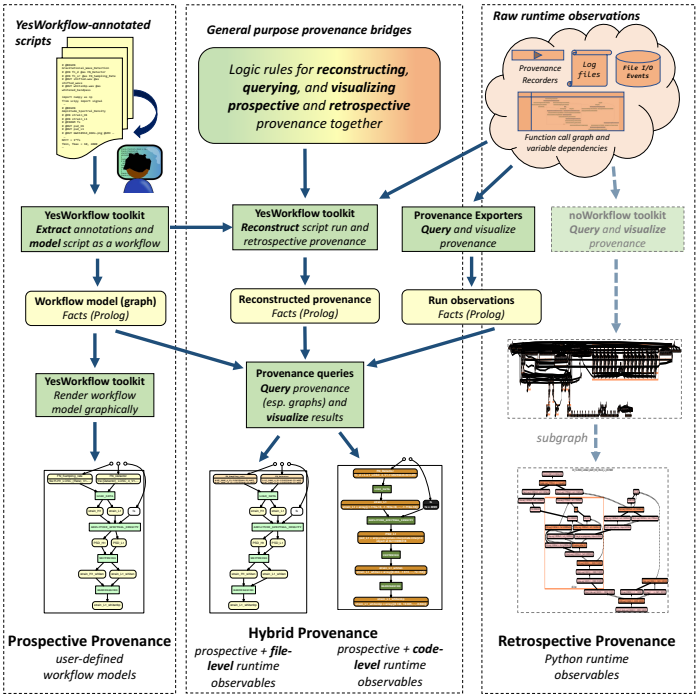

Figure 1. Retrospective provenance (left) and prospective provenance (right) are combined to create different hybrid provenance products (center).

YW-RM Bridge. The, DataONE RunMangers, i.e., R and MATLAB clients, can record a script run and populate the collected file-level provenance in a SQLite database, consisting of three main tables: an execution metadata table, a file metadata table, and a tag table. This information can be exported as YAML file, which in turn can be joined with the YW model to obtain a hybrid provenance graph with file level runtime observables. The DataONE toolbox can also be used to publish data products together with their provenance to a repository that implements the DataONE service API. For example, individual script runs will generate a DataONE metadata package with associated retrospective provenance. After the data package has been indexed by a coordinating node, the logical connection can be viewed and explored via DataONE search[5]. This facilitates data sharing in the community with proper attribution of results transitively across generations of derived data products, i.e., the user can publish a DataONE-compliant data package in OAI-ORE[6] format (including the ProvONE provenance document, the script itself, and its YW-generated workflow view) to a member node within the DataONE federated network of member node repositories [4].

[5] DataONE Search Site: https://search-sandbox-2.test.dataone.org/#data
[6] Open Archives Initiative Object Reuse and Exchange

YW-NW Bridge. Similar to YW-RM bridge, YW-NW bridge is to integrate the prospective provenance implicit in the script that is exposed from YesWorkflow and the retrospective provenance captured from noWorkflow. Since noWorkflow is a provenance recorder that is capable of capturing runtime observables at both file- and code- level, there are thus two types of YW-NW bridge for hybrid provenance queries. (1) **YW-NW bridge at the file level**: Similar to the YW-RM bridge approach, which by selecting file-level only trace information generated by NW, we export such contents stored in the noWorkflow SQLite database tables into a YAML-based format that can be joined with YW prospective information to reconstruct hybrid provenance that can be queried by YW model, thus providing a simplified YW-NW bridge at file-level. (2) **YW-NW bridge at the code level**: This is also called Yin & Yang approach [3] since it traces data lineage in terms of YW model with the details from NW provenance at the program variable-level. Figure 2 uses a YW prospective graph to conceptually illustrate this bridge. Green boxes represent computational steps and yellow for input/output files in between. YW-NW bridge looks for the corresponding variables defined in YW annotation and those in NW, generates complementary relationship-information tables between variables in noWorkflow and workflow program blocks expressed in YW and vice versa, and finally builds up connections between user-defined data / variables and their runtime values by combining information gathered by NW and YW. With this bridge, we are able to answer hybrid provenance queries of specific data such as "Given the output value of a variable in an execution of script, what are values of the inputs of a scientific operation or computational step?", as well as build data lineage visualization that combines both prospective and retrospective provenance information, and yet preserve the simplicity of YW graphs. The concrete example will be demonstrated in the LIGO use case.

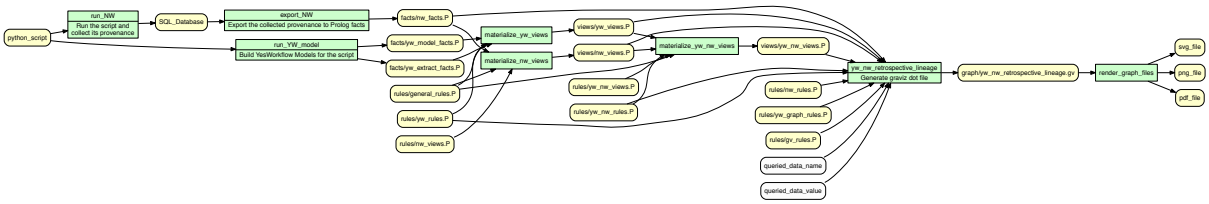

Figure 2. YW-NW bridge model: Use YesWorkflow prospective graph to conceptually visualize the YW-NW bridge at the code level.

4. Example Use Cases

In this section we will demonstrate four use cases - the four script-based workflow use cases that cover multiple disciplines (astrophysics, climate science, biodiversity data curation, and social networks), are implemented in different programming languages (Python, R, and MATLAB) and query languages (SQLite and Prolog-xsb), and highlight the usefulness of diverse forms of retrospective provenance when combined with prospective provenance. In a nutshell, we demonstrate three retrospective provenance capturing systems: a lightweight provenance capturing system YW-recon, a MATLAB provenance capturing system DataONE RunManager, and a Python provenance capturing system noWorkflow. The four use cases are summarized as below:

- C3C4 use case: A MATLAB script in climate science, whose provenance information is collected combining YW and MATLAB RM at the file level.
- Twitter use case: A Python script in social media analytics, whose provenance information is captured combining YW and NW at the file level.

- LIGO use case: A Python script in Astrophysics, whose provenance information[7] is obtained combining YW and NW at the program variable level.
- SPNHC use case: A Python script in biodiversity data curation, whose provenance information is attained combining YW and its extended log features.

For each example, by using the YW prospective provenance modelling, a researcher can visualize a YW-annotated script as a ProvONE-compliant workflow graph. However, data dependencies between script outputs and inputs are hard to trace, especially for complex scripts, particularly at the source code level, not even to mention in YW-rendered graph form. In [6, 10] we demonstrated how to query different types of provenance graphs, i.e., prospective, retrospective, and hybrid graphs. We demonstrated the prospective provenance created by YW and the retrospective provenance collected by various provenance capturing systems or a combination of several together to answer queries that cannot be answered solely by prospective provenance or retrospective provenance. In a more general sense, the following prospective, retrospective, and hybrid queries can be supported for all our use cases.

- Q1: Render everything upstream of a given data product D, where D can be any one (output or intermediate) data element of the YW model of the script.
- Q2: List the script inputs that are upstream of a given data product D.
- Q3: Render the workflow graph downstream of a particular script input.
- Q4: List the outputs that depend on / downstream of a particular script input.
- Q5: Render recon workflow graph upstream of a given data product D.
- Q6: Render recon workflow graph with all observables.

Use Case: Twitter Sentiment Analysis. Sentiment analysis has been very popular in social media analytics over the past few years. In this example, we describe a simple script implemented in Python that uses the NLTK[8] library to assign sentiment scores to Tweets. The script is commented using YW annotations and the conceptual workflow (prospective) graph is visualized by YW in Figure 3(a), while Figure 3(b) shows the downstream prospective influence subgraph for the data element "PositiveCount". The green box represents a computation step and the yellow represents a data element. Both graph queries demonstrated in Figure 3. could be retrieved by querying Yesworkflow prospective provenance based on Prolog without executing the Python script. Also the script is further investigated for more advanced retrospective and hybrid queries, the detail of which has been checked in on the GitHub repository[9].

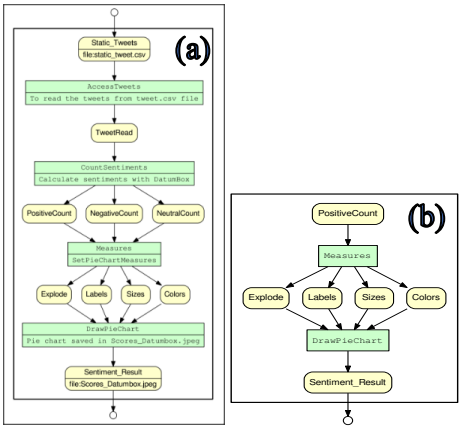

Figure 3. Twitter example (sub)workflows: (a) complete prospective graph; (b) Q3 prospective graph: prospective graph (based on YW-perspective provenance) that renders downstream of the data element "PositiveCount".

Use Case: C3C4. In this section, we use a DataONE example to demonstrate hybrid provenance and hybrid queries. The example script (in MATLAB) produces Carbon

[7] Provenance information consists of prospective provenance and retrospective provenance.
[8] Sentiment analysis: http://www.nltk.org/howto/sentiment.html
[9] Twitter use case GitHub repo: https://github.com/yesworkflow-org/yw-idcc-17/tree/master/examples/Twitter

3/Carbon 4 (C3/C4) soil maps for North America using average rain and air temperature monthly data from year 2000 to 2010. An earth scientist uses YW tool to mark up the script and exposes the underlying prospective provenance that is inherent in the script as shown in Figure 4(a). It is noted that *URI template* is used in input and output data files to specify the file name and path metadata. For example, the data "*mean_precip*" includes three template variables in curly braces It is easy to see from Figure 4(a) that there are three inputs data ("*mean_precip*", "*mean_airtemp*", and "*SYNMAP_land_cover_map_data*") and three output data elements ("*C3_fraction_data*", "*C4_fraction_data*", and "*Grass_fraction_data*"), respectively. Then, a provenance query can be asked based on the prospective provenance harvested by YW: Can you show me the upstream workflow graph for a given data element, say

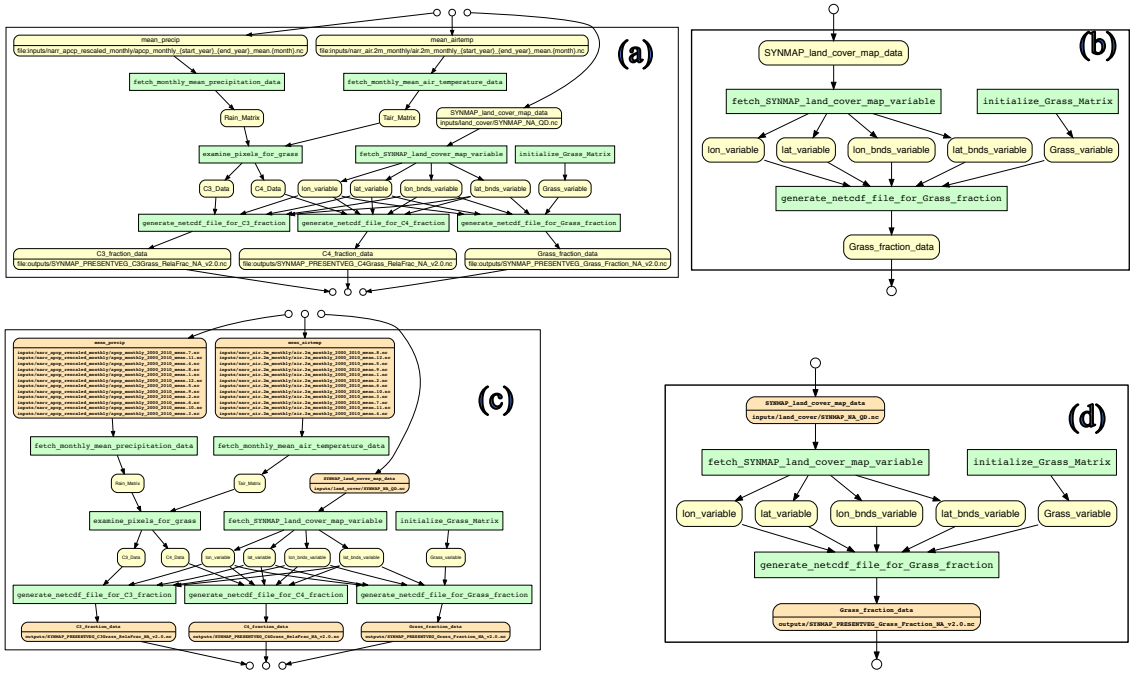

Figure 4. C3C4 example (sub)workflows: (a) complete prospective provenance graph (with YW-URI templates); (b) Q1 prospective graph: prospective graph (based on YW-perspective provenance) that renders upstream of the data element "*Grass_fraction_data*"; (c) Q6 hybrid graph: **complete augmented prospective graph with file-level runtime observables** (based on **YW-URI-RM at file level**); (d) Q5 hybrid graph: **augmented prospective graph with file-level runtime observables** (based **YW-URI-RM at file level)** that renders upstream of the data element "*Grass_fraction_data*".

"*Grass_fraction_data*"? The answer to this query is plotted in Figure 4(b), which indicates that the "*Grass_fraction_data*" ***doesn't depend on*** the first two inputs ("*mean_precip*" and "*mean_airtemp*") *but* ***possibly depends on*** only one input data -- "*SYNMAP_land_cover_map_data*", instead of all three of them. This data lineage analysis that uses techniques based on file-level analysis slicing to represent the provenance of database queries makes it possible to show how (part of) the output data *does not definitely depend on* (parts of) its input. On the other hand, by using the MATLAB RM tool to record a run of this script, the user can not only obtain the expected execution results, but can also capture the file-level retrospective provenance. Then, the hybrid provenance can be derived by joining YW prospective provenance and RunManager retrospective provenance bridge with YW URI templates (YW-URI-RM at file level). After obtaining the derived hybrid provenance information, we can ask hybrid queries. For example, (1) Show the complete reconstructed augmented prospective graph with all file-level runtime observables; (2) Given a data item

"*Grass_fraction_data*", show the reconstructed upstream augmented provenance graph in context of the YW workflow graph. The generated reconstructed (sub)graphs containing derived hybrid provenance are shown in Figure 4(c) and Figure 4(d), respectively. From Figure 4(c), we can see that when the script is running, it actually reads twenty-four (12*2) data files containing rain/air temperature data files corresponding to every month from the year 2000 to 2010, plus the additional land map file read from the script, resulting in total of 25 input data files rather than 3 from the YW (prospective) workflow (Figure 4(a)). Then, the script writes three files to the outputs folder.

Use Case: LIGO. LIGO Open Science Center detected gravitational waves from a binary black hole merger on September 14 2015, and called the event GW150914 [22]. With strain time series data from GW150914 collected by Detector H1 from the LIGO Hanford Observatory and Detector L1 from LIGO Livingston Observatory, electronical engineers write a Python script to process signal processing tasks that eliminate the disturbances from local source and visualize the gravitational wave in spectrogram and audio form. Figure 5(a) depicts the YW subgraph upstream of variable "strain_L1_whitenbp" in YW model. Figure 5(b) and 5(c) both demonstrate the reconstructed workflow graph upstream of the same data, whose differences lie in that the former was implemented via YW-URI template at the file-level while the latter YW-NW bridge at the code level. In the graph model of YW-NW bridge at program variable level (Figure 5(c)), the black box represents a parameter variable associated with its value, and the orange box for an input/output data associated with its value. YW-NW bridge approach can not only enable queries that can be answered by YW, but can also answer queries for runtime data information that can't be answered solely by YW. For example, show variables and their values that are the inputs of the Bandpassing workflow step, and emits output "*strain_L1_whitenbp*" with value of [8.184, …, -0.684]. From the prospective provenance graph of YW (Figure 5(a)), both data products "*strain_H1_whiten*" and "*strain_L1_whiten*" appear as inputs flowing into the computational step Bandpassing from which output "*strain_L1_whitenbp*". However from the graph model of YW-NW bridge at program variable-level in Figure 5(c), ***only*** *"strain_L1_whiten" is being bandpassed* with value [8.494, ..., 72.156] to produce the desired "*strain_L1_whitenbp*", which is also self-explanatory from their data names. This means that "*strain_H1_whiten*" is being passed to and used in bandpassing, but is irrelevant to the output of "strain_L1_whitenbp". From another point of view, we also noticed from YW-NW bridge at file-level (Figure 5(b)), that two files are being detected as input for "FN_Detector" since the augmented prospective graph lists all actual inputs of the program that correspond to data variables, whereas only one of the two input files for L1 detector is consumed and thus shown within YW-NW bridge at code level (Figure 5(c)). Not only the YW-NW bridge at the code level records the lineage of the queried data is recorded, but captures the data dependency to the actual input file.

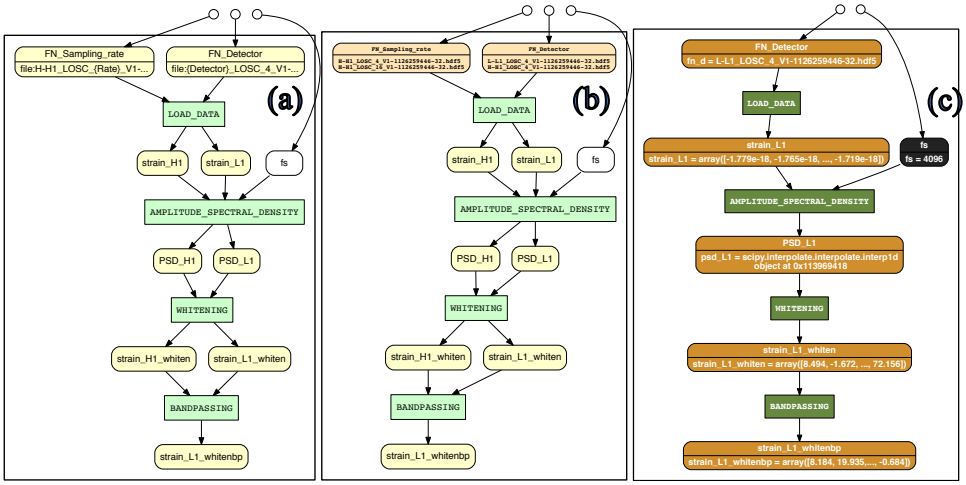

Figure 5. LIGO example (sub)workflows: (a) Q1 prospective graph: prospective graph with URI templates (based on YW-perspective provenance) that renders upstream of the data element "*strain_L1_whitenbp*"; (b) Q5 hybrid graph: **augmented prospective graph with file-level runtime observables** (based on **YW-recon-URI at file level)** that renders upstream of the data element "*strain_L1_whitenbp*"; (c) Q5 hybrid graph**: retrospective graph with all (both file- and code-level) runtime observables and data dependencies** (based on **YW-NW bridge at code level**) that renders upstream of the data element "*strain_L1_whitenbp*".

Therefore from LIGO and C3C4 examples, we can see that the YW reconstructed file-level runtime observables alone (1) can determine the definite independency between data products; (2) may or may not precisely define the data dependencies. In contrast, the YW-NW bridge can generate code-level runtime information, which once be integrated with YesWorkflow, can capture the correct data lineage and dependency relationships that supplement retrospective data information for YesWorkflow models.

Use Case: SPNHC

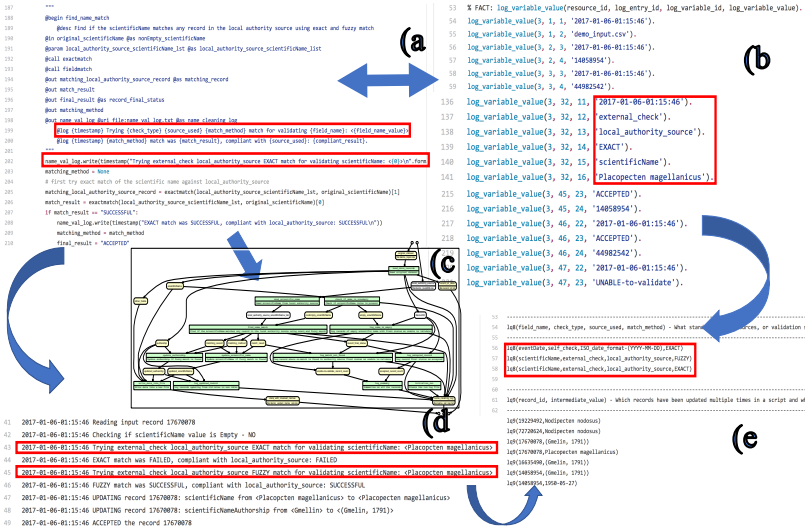

Figure 6. YW-log implementation for the SPNHC biodiversity data cleaning use case: (a) YW-annotated (@out, @uri, and @log) script that declare log file, file path, and log template; (b) *YW-recon* log_variable_value facts; (c) YW-rendered prospective graph (d) the log file written by the script (e) log queries.

In this use case, we demonstrate how the YW toolkit developed by the Kurator[10] team facilitates exploration of the results of running a simple data cleaning script written in Python (Figure 6(a)) on a biodiversity dataset. This script takes as input a CSV file containing occurrence records represented using Darwin Core terms; in successive steps validates the various fields (scientific name, authorship, and event date) of each input record; replaces nonstandard or incorrect values for particular fields when the meanings or intents of these input values are

[10] opensource.ncsa.illinois.edu/confluence/display/KURATOR/Kurator+Project+Home

unambiguous; and optionally discards records missing values for essential fields or containing incorrect or ambiguous values for which enhancements cannot be proposed. The cleaned data set ultimately produced by the script is written to another CSV file. Each of the data validation steps in the script write key information about their operations into simple log files (.txt). On the one hand, by analyzing the YW annotations in a script, YesWorkflow can render a prospective provenance graph of a script to be executed (Figure 6(c)). On the other hand, YW retrospective provenance reconstructions based on *URI* and log template matching can assist interpreting and querying log files in terms of the structure of the script itself. YW can answer questions about the output data set, the script run, and the data validation, field updates, and record removal events that occurred during the data cleaning process. Using log files easily written by such scripts, YesWorkflow can reveal which fields in particular records were marked as suspect and why, which of these fields were corrected, and what records were removed. YesWorkflow depends on special comments in the script, and thus these annotations enable interpreting and querying log files in terms of the structure of the script itself. Figure 6(d) and 6(e) depicts the log file content and log query output using URI template and extended log feature.

5. Conclusions and Future Work

The challenges we try to conquer in this paper include: (1) Most computational analyses and workflows are conducted using scripts in different programming languages, such as Python, R, MATLAB, and etc. (2) Retrospective provenance observables, from DataONE RunManager (file-level), or noWorkflow (Python code-level) only yield isolated fragments of the overall data lineage and processing history. (3) Prospective provenance could be used to link and contextualize fragments into a meaningful and comprehensible workflow, but script alone do not reveal the underlying workflow graph. (4) Provenance (like other metadata) appears to be rarely actionable or immediately useful for those who are expected to provide it ("provenance for others"). The approach we propose for hybrid provenance in this paper consists of three steps: (1) Simple YesWorkflow (YW) annotations allow users to explicitly reveal the workflow (prospective provenance) that was originally implicit in scripts. (2) Prospective provenance queries expose and test data dependencies at the workflow level. (3) Hybrid provenance queries situate runtime observables (retrospective provenance) in the overall workflow, yielding meaningful knowledge artifacts. Our proposed approach, integrate comprehensible workflow graphs and customizable provenance reports for script runs, along with data and code in scientific studies ("provenance for self"). Using the example scripts as a testbed, we show how the dual views of retrospective and prospective provenance can both reveal the overall story of a script run, and also the detailed history of particular script products. An earlier hands-on demonstration illustrating our approach is available via an open source repository [6].

As to future work, besides the single script run provenance, we are also interested in multi-run and /or multi-script executions, the former of which means to execute the same script for multiple times in order to compare among different input settings or to seek optimal output result; while the latter scenario occurs in big computation workflows run that could last days, weeks, or even months which usually consist of more than one script or a cascade of subroutines of multiple scripts using different input argument(s) and / or parameters configured at each setting. Under such circumstance, some intermediate data files could be generated and consumed implicitly within such a long chain reaction, which usually play a key role in determining the final model output

however easily get ignored if not enough attention were paid. In this complex case, we are interested in what kind of provenance information could be captured, queried or visualized and at which levels of granularity (e.g., at the script level, data level, field level, record level, file level, function level, provided that it has been configured at each). In order to reveal data lineage, we might again need to use a combination of YW and other provenance tracking tools (e.g., DataONE RunManager, NW) for exposing prospective, retrospective and hybrid provenance. We will demonstrate our preliminary results in the demo session of this IDCC conference, showing the new capabilities described above.

Another part of future work, we want to generate and query ProvONE-compatible RDF representations of YW annotations, workflow models, and retrospective provenance so that ProvONE compatible vocabulary extensions may be used in YW in the future via the mapping between YW and ProvONE data model.

Acknowledgments. Work supported in part by NSF awards DBI-1356751 (Kurator), ACI-1430508 (DataONE), SMA-1637155 (SKOPE), and ACI-1541450 (Whole Tale).

References

[1] T. McPhillips, T. Song, T. Kolisnik, S. Aulenbach, K. Belhajjame, R.K. Bocinsky, Y. Cao, J. Cheney, F. Chirigati, S. Dey, J. Freire, C. Jones, J. Hanken, K.W. Kintigh, T.A. Kohler, D. Koop, J.A. Macklin, P. Missier, M. Schildhauer, C. Schwalm, Y. Wei, M. Bieda, B. Ludäscher (2015). YesWorkflow: A User-Oriented, Language-Independent Tool for Recovering Workflow Information from Scripts. International Journal of Digital Curation 10, 298-313.

[2] T. McPhillips, S. Bowers, K. Belhajjame, B. Ludäscher (2015). Retrospective Provenance Without a Runtime Provenance Recorder. 7th USENIX Workshop on the Theory and Practice of Provenance (TaPP 2015).

[3] J. F. Pimentel, S. C. Dey, T. M. McPhillips, K. Belhajjame, D. Koop, L. Murta, V. Braganholo, B. Ludäscher. Yin & Yang: Demonstrating Complementary Provenance from noWorkflow & YesWorkflow. IPAW 2016: 161-165.

[4] Y. Cao, C. Jones, V. C.-Vicenttín, M. B. Jones, B. Ludäscher, T. M. McPhillips, P. Missier, C. R. Schwalm, P. Slaughter, D. Vieglais, L. Walker, Y. Wei. DataONE: A Data Federation with Provenance Support. IPAW 2016: 230-234.

[5] L. Murta, V. Braganholo, F. Chirigati, D. Koop, J. Freire. noWorkflow: Capturing and Analyzing Provenance of Scripts. IPAW 2014: 71-83.

[6] Y. Cao, D. Vu, Q. Wang, Q. Zhang, P. Thavasimani, T. McPhillips, P. Missier, B. Ludäscher (2016). YW query demo site on Github: https://github.com/idaks/dataone-ahm-2016-poster.

[7] Bowers, Shawn, Timothy McPhillips, and Bertram Ludäscher. "Declarative Rules for Inferring Fine-Grained Data Provenance from Scientific Workflow Execution Traces." In Provenance and Annotation of Data and Processes, 82–96. Springer, Berlin, Heidelberg, 2012. doi:10.1007/978-3-642-34222-6_7.

[8] Cuevas-Vicenttín, V., Ludäscher, B., Missier, P., Belhajjame, K., Chirigati, F., Wei, Y., ... & Altintas, I. ProvONE: A PROV Extension Data Model for Scientific Workflow Provenance (2015). http://jenkins-1.dataone.org/jenkins/view/Documentation%20Projects/job/ProvONE-Documentation-trunk/ws/provenance/ProvONE/v1/provone.html.

[9] Bavoil, L., Callahan, S. P., Crossno, P. J., Freire, J., Scheidegger, C. E., Silva, C. T. & Vo, H. T. (2005). VisTrails: Enabling interactive multiple-view visualizations. In Visualization 2005 (VIS '05) (pp. 135–142). IEEE. doi:10.1109/ VISUAL.2005.1532788

[10] Q. Zhang, Y. Cao, Q. Wang, D. Vu, P. Thavasimani, T. McPhillips, P. Missier, B. Ludäscher (2017). yw-idcc-17 hybrid query demo site on GitHub: https://github.com/yesworkflow-org/yw-idcc-17.

[11] Lerner, B. S., & Boose, E. R. (2014). Collecting provenance in an interactive scripting environment. In Workshop on the Theory and Practice of Provenance (TaPP), Cologne, Germany.

[12] I. Altintas, O. Barney, and E. Jaeger-Frank. Provenance collection support in the Kepler scientific workflow system. In IPAW, 2006.

[13] S. Bowers, T. McPhillips, S. Riddle, M. K. Anand, and B. Ludäscher. Kepler/pPOD: Scientific workflow and provenance support for assembling the tree of life. In IPAW, 2008.

[14] P. Missier, K. Belhajjame, J. Zhao, M. Roos, and C. Goble. Data lineage model for Taverna workflows with lightweight annotation requirements. In IPAW, 2008.

[15] K. Wolstencroft, R. Haines, D. Fellows, A. Williams, D. Withers, S. Owen, S. Soiland-Reyes, I. Dunlop, A. Ne- nadic, P. Fisher, et al. The Taverna workflow suite: designing and executing workflows of Web Services on the desktop, web or in the cloud. Nucleic Acids Research, 2013.

[16] T. Oinn, M. Addis, J. Ferris, D. Marvin, M. Senger, M. Greenwood, T. Carver, K. Glover, M. R. Pocock, A. Wipat, P. Li, Taverna: A Tool for the Composition and Enactment of Bioinformatics Workflows, Bioinformatics 20 (17) (2004) 3045–3054.

[17] Deelman, E., Singh, G., Su, M. H., Blythe, J., Gil, Y., Kesselman, C., ... & Laity, A. (2005). Pegasus: A framework for mapping complex scientific workflows onto distributed systems. Scientific Programming, 13(3), 219-237.

[18] Goecks, J., Nekrutenko, A., & Taylor, J. (2010). Galaxy: a comprehensive approach for supporting accessible, reproducible, and transparent computational research in the life sciences. Genome biology, 11(8), 1.

[19] Ludäscher, B., Altintas, I., Berkley, C., Higgins, D., Jaeger, E., Jones, M., . . . Zhao, Y. (2006). Scientific workflow management and the Kepler system. Concurrency and Computation: Practice and Experience, 18(10), 1039–1065. doi.org/10.1002/cpe.994

[20] W3C PROV-O: The PROV Ontology. https://www.w3.org/TR/prov-o/

[21] Tsai, Y., McPhillips, S. E., González, A., McPhillips, T. M., Zinn, D., Cohen, A. E., . . . Soltis, S. M. (2013). Autodrug: fully automated macromolecular crystallography workflows for fragment-based drug discovery. Acta Crystallographica Section D: Biological Crystallography, 69(5), 796–803. doi:10.1107/S0907444913001984

[22] LIGO Open Science Center: Signal Processing with GW150914 Open Data. https://losc.ligo.org/events/GW150914/

[23] Zhao, Y., Wilde, M., & Foster, I. (2006, May). Applying the virtual data provenance model. In International Provenance and Annotation Workshop (pp. 148-161). Springer Berlin Heidelberg.

[24] Freire, J., Fuhr, N., & Rauber, A. (2016). Reproducibility of Data-Oriented Experiments in e-Science (Dagstuhl Seminar 16041). In Dagstuhl Reports (Vol. 6, No. 1). Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik.

[25] J. Freire, D. Koop, F. S. Chirigati, and C. T. Silva. Reproducibility using vistrails. Implementing Reproducible Research, page 33, 2014.

[26] C. Gandrud. Reproducible Research with R and R Studio. CRC Press, 2013.

[27] Tilmes, C., Yesha, Y., & Halem, M. (2010). Tracking provenance of earth science data. Earth Science Informatics, 3(1-2), 59-65.

Appendix

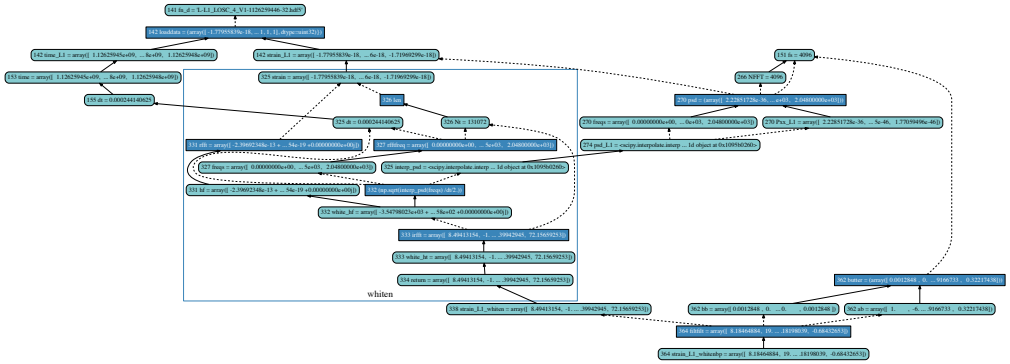

Figure A1. noWorkflow subgraph query example: show the filtered (upstream) lineage graph for data element `Strain_L1_whitenbp`

The Python-based noWorkflow (NW) Provenance Recorder

Another runtime provenance recorder is noWorkflow [5]. NW is a command line tool written in Python that doesn't need to modify scripts to include annotations but instead exploits a Python profiling library to collect different types of provenance information without version control system about contents of function call stacks and variable assignments at runtime, leveraging well-known abstract syntax tree analysis, profiling, and reflection techniques [5]. NW transparently captures three types of provenance of Python scripts – definition (the structure of the script that corresponds to prospective provenance), deployment (execution environment), and execution (execution log for the script that corresponds to retrospective provenance). NW captures provenance not only at the operating system level[11], but also at the code-level[12] as well as at the file level[13]. All the provenance information collected by NW is stored in SQLite database, which is exported as Prolog facts and rules for efficient provenance query capabilities that follow the same structure of the relational tables as in SQLite. Therefore, NW supports different classes of provenance analysis: (1) visualizing provenance overview of a trail[14]; (2) diff-based comparison analysis between different trails / runs; and (3) provenance queries of a given trial based on Prolog. However fine-grained provenance may overwhelm scientists for large volumes of low-level provenance data and/or may lead to significant execution overhead and visualization overload. As shown in [3], by combining YW and NW provenance, hybrid provenance may achieve the "best of both worlds".

[11] NW captures provenance at the operating system level, which monitors system calls and tracks processes and data dependencies between these processes.
[12] NW captures provenance at the code level, including control flow information, library dependencies, function and variable dependencies, etc.
[13] NW captures provenance at the file level, including general data dependencies from file reads and writes, as well as changes in files (e.g., script modifications, environment variable changes, and module updates, etc.).
[14] In noWorkflow, each execution of a script is referred to as a trial [5].