noWorkflow: a Tool for Collecting, Analyzing, and Managing Provenance from Python Scripts

João Felipe Pimentel[1] Leonardo Murta [1] Vanessa Braganholo[1] Juliana Freire[2]
[1]Universidade Federal Fluminense [2]New York University
{jpimentel,leomurta,vanessa}@ic.uff.br juliana.freire@nyu.edu

ABSTRACT

We present noWorkflow, an open-source tool that systematically and transparently collects provenance from Python scripts, including data about the script execution and how the script evolves over time. During the demo, we will show how noWorkflow collects and manages provenance, as well as how it supports the analysis of computational experiments. We will also encourage attendees to use noWorkflow for their own scripts.

1. INTRODUCTION

Provenance helps users to interpret and reason about the results of computational processes [6]. For scientific experiments, it captures all computational steps and data that contribute to the output, thus enabling scientists to review these steps, input and intermediate data, and assess the quality of the derived results. For instance, if an experiment leads inconclusive results, the provenance data may help understanding the reasons. Moreover, when running several trials of a given experiment, scientists can cache intermediate data to avoid re-computing expensive operations [7]. Provenance also enables reproducibility by providing information regarding not only the computational steps but also the libraries and environment dependencies [3]. With such information, users can match the libraries and environment configurations to reduce external influences and reproduce experiments under similar conditions. After reproducing an experiment, provenance can be used to verify whether the same steps are executed, or if there are differences between the original run and the re-execution. Finally, scientists can use provenance to manage the evolution of experiments: they can create snapshots, restore specific versions of the experiment, libraries, and input data. In experiments that involve parameter exploration, scientists can create versions with each set of parameters, and use provenance to keep track of inputs and outputs [10].

Several approaches have been proposed to capture the provenance of computational processes. Tools that track provenance at the operating system level [8, 11] have two major benefits: they are general and language independent, and provenance can be automatically collected without requiring user intervention. However, the captured information can be hard to reason and to connect to the actual semantics of the experiments. Workflow management systems (WFMS) require scientists to specify their experiment as workflows [2, 19]. These tools support different levels of abstraction and when workflows are properly designed, the provenance closely matches experiment semantics. However, WFMS often require a steep learning curve and high adoption costs [17]. Furthermore, they lack the flexibility of general-purpose languages. For these reasons, many scientists still use scripts [5].

Tools have also been developed to track provenance from scripts [1, 9, 12, 18]. While some tools collect provenance automatically [12, 18], most focus on a single trial (i.e., a single execution of an experiment). As a result, these tools do not collect the information required to check for repeatability, do not support data re-use, and they are unable to manage experiment evolution. In addition to automatically collecting provenance from the execution of Python scripts, *noWorkflow* [12] addresses these limitations by tracking their history and evolution [14]. Thus, users can analyze multiple trials, compare them, and understand their history. In previous work, noWorkflow was extended to collect provenance and run analyses on interactive notebooks [16] and it was also combined to YesWorkflow [9] to link prospective provenance collected by YesWorkflow with retrospective provenance collected by noWorkflow [13].

In our demonstration, we will use real experiments as well as scripts provided by attendees and walk them through the process of provenance collection, analysis, and management supported by noWorkflow. We will show how provenance can be collected at different levels of granularity and showcase the different operations and visual representations noWorkflow provides to help users query and visualize provenance information.

2. OVERVIEW OF NOWORKFLOW

Collecting provenance of scripts is challenging. First, one must select the appropriate level of granularity. While coarse-grained provenance may hide important data, fine-grained provenance may overwhelm users. Moreover, scripts can encode control flow, cycles, and other structures that make it difficult to identify which parts of the scripts contributed to the generation of a given data product. Finally, scripts run outside controlled environments. Thus, it is hard to make assumptions based only on the results of scripts, as the environment may interfere with the execution.

Proceedings of the VLDB Endowment, Vol. 10, No. 12
Copyright 2017 VLDB Endowment 2150-8097/17/08.

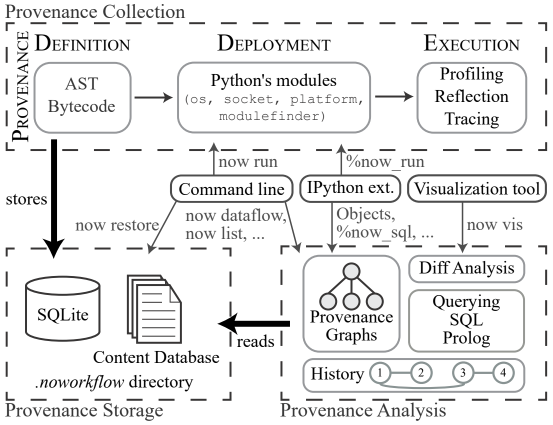

Figure 1: Architecture of noWorkflow.

noWorkflow addresses these challenges by collecting *definition*, *deployment*, and *execution* provenance. *Definition provenance* represents the structure of the script, including function definitions, their arguments, function calls, and other static data. *Deployment provenance* represents the execution environment, including information about the operating system, environment variables, and libraries on which the script depends. Finally, *execution provenance* represents the execution log for the script [12].

noWorkflow supports different techniques for collecting execution provenance at different levels of granularity. These techniques deal with the existence of control flows and cycles on scripts. Moreover, by collecting deployment provenance, noWorkflow is able to detect environment changes that are external to the script.

The architecture of noWorkflow has three key components, as shown in Figure 1. The *Provenance Collection* module collects provenance from scripts and stores it using the *Provenance Storage* module. The *Provenance Analysis* module reads data from the *Provenance Storage* and presents it to users in different ways. Users can interact with noWorkflow through three interfaces: command line, an IPython extension that interacts with Jupyter Notebook, and a web-based visualization tool. In addition to provenance collection and analysis, noWorkflow also collects the evolution history of experiments, allowing users to restore old files and to manage their execution.

3. DEMONSTRATION

In our demonstration scenario, a user receives a request from her collaborator to check if the precipitation in Rio de Janeiro remains constant across years. To verify this hypothesis, she collects data from a meteorological database and writes a script to process the data and produce a image for comparison. She starts the experiment with data from 2013 and 2014 and produces the script presented in Figure 2.

3.1 Provenance Collection

Running noWorkflow is as simple as running a Python script: instead of invoking `python experiment.py`, she invokes `now run experiment.py`. noWorkflow is able to run the very same Python script and produce the same results without modifications. However, instead of just running it, noWorkflow first generates a sequential trial identification number. Then, it collects the definition provenance and deployment provenance, and when it executes the script, it collects the execution provenance. As this is the first trial, the trial number will be 1. She can now use this number to reference the collected data for this trial. noWorkflow stores the collected provenance on disk in a directory named `.noworkflow` inside the script directory. This directory contains both a relational database for structured data and a content database for file-based data (see the Provenance Storage module of Figure 1). noWorkflow copies all accessed files, modules, and scripts to the content database during the execution of the scripts, and names them after the SHA1 hash codes of their content. These SHA1 hash codes link the content stored in the databases and avoid the duplicated storage of identical files.

```
import numpy as numpy
from precipitation import read, sum_by_month
from precipitation import create_bargraph

months = np.arange(12) + 1
d13, d14 = read("p13.dat"), read("p14.dat")
prec13 = sum_by_month(d13, months)
prec14 = sum_by_month(d14, months)

create_bargraph("out.png", months,
        ["2013", "2014"], prec13, prec14)
```

Figure 2: First script version.

noWorkflow collects the *definition provenance* by analyzing the Abstract Syntax Tree (AST) and the Python bytecode of the script [15]. It stores the script file and function definitions in the content database, and function names, calls, parameters, and global variables in the relational database. For this experiment, the content of 'experiment.py' is stored in the content database, while its hash code together with the function calls (i.e., `arange`, `read`, `sum_by_month`, and `create_bargraph`) are stored in the relational database.

noWorkflow collects two different types of deployment provenance: environment and library dependencies. As part of the environment, it collects operating system information (e.g., Ubuntu 16.04), hostname, information about the machine architecture (e.g., x86_64), Python version (e.g., 3.5.2), and environment variables. This data is stored in the relational database. noWorkflow also collects the transitive closure of all library dependencies together with their versions. In our example, it collects 'numpy' in version 1.11.3, 'precipitation.py' in version 1.1.0 (as it was declared by a __VERSION__ variable), and 'matplotlib' in version 2.0.0, among other internal modules. Note that 'matplotlib' was imported by 'precipitation.py'. noWorkflow stores the libraries in the content database, together with their names, versions, and hash codes in the relational database.

For execution provenance, noWorkflow stores copies of input and output files in the content database before and after reading or writing on them. Thus, it stores copies of 'p13.dat', 'p14.dat', and 'out.png' in the content database. In addition to the accessed files, noWorkflow supports collecting the execution flow at two different granularities. By default, noWorkflow collects provenance at a coarse granularity. It defines a Python Profiler that collects function activations (i.e., executed function calls), global variables, parameters, and return values. To collect provenance at a finer granularity, the user can issue the command `now run -e Tracker experiment.py`. Besides function activations, noWorkflow will also capture variable attributions, loop definitions, and other variable dependencies [15]. In this case, it combines the Profiler with a Tracer to store variable values

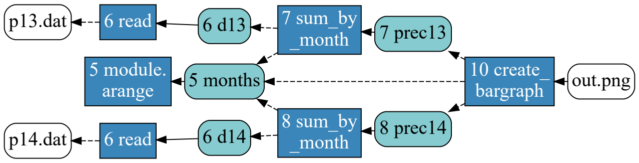

Figure 3: Trial 1 dataflow.

for each line and variable dependencies. A Tracer is commonly used for developing debuggers in Python. However, different from debuggers, noWorkflow stores the data values to assist users in future analyses. noWorkflow stores all data related to the execution flow in the relational database.

As a concrete example, consider line 5 of Figure 2. By default, noWorkflow captures that `np.arange(12)` returns a numpy array with values $[0, 1, \ldots, 10, 11]$. At fine-granularity mode, noWorkflow also collects the value of `months` as an array with values $[1, 2, \ldots, 11, 12]$. In both cases, noWorkflow collects the time of the execution and the function activation duration.

Provenance size may grow considerably if it is captured at a fine granularity or even at a coarse granularity, if script contains large loops. In order to avoid these problems, users can (i) limit the maximum collection depth of noWorkflow stack; (ii) write computational intensive functions in external files, since noWorkflow only collects execution provenance from main scripts, by default; or (iii) prepend an underscore to variables and function names, indicating which elements should not be collected.

3.2 Provenance Analysis

After collecting the provenance, noWorkflow offers multiple commands for provenance analysis. The user can run `now show 1` to obtain more details from trial 1 (textually). Users can also inspect individual modules, function definitions, environment variables, or function activations. This information can be visualized using the command `now dataflow 1 | dot -Tpng -o p1.png`, which produces a dataflow graph, as shown in Figure 3. In this figure, nodes with rounded corners represent data, white nodes represent files, light blue rectangles represent variables, and dark blue rectangles nodes represent function calls.

Since all provenance data, with the exception of file content, is stored in a relational database, the user can query the provenance using SQL. However, to enable complex, transitive closure queries, for which the support in some database systems is limited [4], noWorkflow also exports the trial provenance as Prolog facts using the command `now export 1`. Optionally, noWorkflow can export pre-defined Prolog rules that define common provenance queries together with the Prolog facts. For instance, the rule `access_influence(1, File, 'out.png')` indicates which files may have influenced the generation of 'out.png' in trial 1. In this case, the result is 'p13.dat' and 'p14.dat'. For obtaining a graphical representation of both SQL and Prolog schemas, users can use the command `now schema [sql,prolog]`.

After analyzing the experiment results, the user realizes that there was a drought in 2014 and decides to check if the precipitation remains constant when there is no drought. Thus, she adapts the script to include data from 2012. After running the experiment, she wants to compare the trials and present the differences to her collaborator.

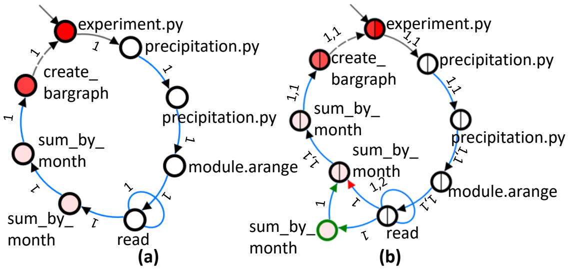

Figure 4: (a) Trial 1 activation graph and (b) Diff of trials 1 and 2.

For a textual trial comparison, she uses the command `now diff 1 2`. Currently, this command just compares basic trial information (e.g., parameters, duration), modules, and environment variables according to specified optional parameters.

However, she also wants to compare the execution provenance, so she uses the noWorkflow web visualization tool. This tool can be activated by the command `now vis` and further accessed on a web browser at http://localhost:5000. The tool presents the history of trials as a graph and allows users to select trial nodes to be visualized in more detail. When the user selects a trial, noWorkflow loads basic trial information, modules, environment variables, accessed files, and an activation graph. By selecting a second trial, the tool compares the first trial to the second one, presenting an activation graph diff [14] and all textual diff information. Different from the dataflow graph presented in Figure 3, activation graphs also work for coarse-grained trials. Figure 4 presents the (a) activation graph of trial 1 and its (b) diff to trial 2. In an activation graph, nodes represent activations and their colors represent their duration in a gradient scale: red represents the slowest activations, and white, the fastest ones. The script is an activation itself and it is indicated by a straight arrow. In this case, 'experiment.py' is the script. In the graph, black arrows represent the start of activations, blue arrows represent a sequence of calls within activations, and dashed arrows represent returns. Note that Figure 4(b) has an extra `sum_by_month` node and an extra read activation number in the loop edge.

3.3 Provenance Management

Now the user decides to change the script to add data from 2015. In the meantime, her collaborator realizes that there are unusual rainy days in the first trial and requests her to rerun the experiment without such days. Since she is using noWorkflow, she can restore the code and data from trial 1 by issuing the command `now restore 1` [14]. Since the user has changed the code, this command creates a backup trial, 3, with the modified script as definition provenance, before restoring the files from trial 1. By default, the command restores the whole trial to the state before its execution, but it supports optional arguments for restoring individual files, including intermediate and output files. Thus, the restore command is useful for trying alternatives on the experiment, for repeating trials, and for looking at old versions of trials. After restoring the trial, the user modifies the script and executes it again without the unusual rainy days.

noWorklow keeps track of the trial derivation history and allows users to visualize this history for understanding what happened to the experiment until it reached the current state. For visualizing the history, she can either run `now`

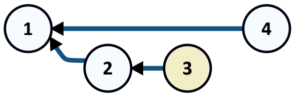

Figure 5: Experiment history with trials as nodes.

history and obtain a textual representation, or load the aforementioned visualization tool. Figure 5 presents the trial history for this demonstration. Note that trial 4 is based on trial 1 and trial 3 appears with a different color that denotes it is a backup trial. If the derivation history is not important, and the user just wants to list all trials with their command lines and durations, she can run `now list`.

Different from standard version control systems, noWorkflow versions are related to trial executions. This allows users to keep the full history of their experiments, keeping track of arguments, input data, output data, and other provenance information.

4. CONCLUSION

In this demonstration paper, we present *noWorkflow*, a tool that automatically collects provenance from Python scripts, without requiring any modification to the script. During the execution of scripts, noWorkflow collects imported modules, environment variables, function calls, file accesses, and, optionally, variables. While it does not collect network activity or database accesses directly, it collects the functions called for such accesses. noWorkflow also tracks the evolution of experiments and allows users to navigate over different versions. noWorkflow provides support for different kinds of provenance analyses through a command line interface, SQL and Prolog queries, and visualizations. Finally, noWorkflow also supports interactive analyses on Jupyter Notebooks.

noWorkflow is under active development. The system is available as open source software at `http://gems-uff.github.io/noworkflow`. Short videos showcasing the tool are available at `http://github.com/gems-uff/noworkflow/wiki/Videos`.

5. ACKNOWLEDGMENTS

We would like to thank CNPq, FAPERJ and the Moore-Sloan Data Science Environment for their financial support for this project. Juliana Freire is supported by the DARPA Memex and D3M programs, and NSF awards ACI- 1640864, CNS-1229185 and CNS-1405927.

6. REFERENCES

[1] C. Bochner, R. Gude, and A. Schreiber. A python library for provenance recording and querying. In *IPAW*, pages 229–240, 2008.

[2] S. P. Callahan, J. Freire, E. Santos, C. E. Scheidegger, C. T. Silva, and H. T. Vo. Managing the Evolution of Dataflows with VisTrails. In *ICDE*, pages 71–71, 2006.

[3] F. Chirigati, D. Shasha, and J. Freire. Reprozip: Using provenance to support computational reproducibility. In *TaPP*, pages 977–980, 2013.

[4] S. Dar and R. Agrawal. Extending Sql with generalized transitive closure. *IEEE Transactions on Knowledge and Data Engineering*, 5(5):799–812, 1993.

[5] S. Dey, K. Belhajjame, D. Koop, M. Raul, and B. Ludäscher. Linking prospective and retrospective provenance in scripts. In *TaPP*, pages 1–7, 2015.

[6] J. Freire, D. Koop, E. Santos, and C. T. Silva. Provenance for computational tasks: A survey. *Computing in Science & Engineering*, 10(3):11–21, 2008.

[7] P. J. Guo and D. Engler. Using automatic persistent memoization to facilitate data analysis scripting. In *ISSTA*, pages 287–297, 2011.

[8] P. J. Guo and M. Seltzer. BUrrIto: Wrapping Your Lab Notebook in Computational Infrastructure. In *TaPP*, volume 12, pages 1–7, 2012.

[9] T. McPhillips, T. Song, T. Kolisnik, S. Aulenbach, K. Belhajjame, K. Bocinsky, Y. Cao, F. Chirigati, S. Dey, J. Freire, et al. YesWorkflow: a user-oriented, language-independent tool for recovering workflow information from scripts. *International Journal of Digital Curation*, 10(1):298–313, 2015.

[10] R. Meyer and K. Obermayer. pypet: A python Toolkit for Data Management of Parameter Explorations. *Frontiers in Neuroinformatics*, 10:1–16, 2016.

[11] K.-K. Muniswamy-Reddy, D. A. Holland, U. Braun, and M. I. Seltzer. Provenance-Aware Storage Systems. In *USENIX ATC*, pages 43–56, 2006.

[12] L. Murta, V. Braganholo, F. Chirigati, D. Koop, and J. Freire. noWorkflow: capturing and analyzing provenance of scripts. In *IPAW*, pages 71–83, 2014.

[13] J. F. Pimentel, S. Dey, T. McPhillips, K. Belhajjame, D. Koop, L. Murta, V. Braganholo, and B. Ludäscher. Yin & Yang: demonstrating complementary provenance from noWorkflow & YesWorkflow. In *IPAW*, pages 161–165, 2016.

[14] J. F. Pimentel, J. Freire, V. Braganholo, and L. Murta. Tracking and analyzing the evolution of provenance from scripts. In *IPAW*, pages 16–28, 2016.

[15] J. F. Pimentel, J. Freire, L. Murta, and V. Braganholo. Fine-grained provenance collection over scripts through program slicing. In *IPAW*, pages 199–203, 2016.

[16] J. F. N. Pimentel, V. Braganholo, L. Murta, and J. Freire. Collecting and analyzing provenance on interactive notebooks: when IPython meets noWorkflow. In *TaPP*, pages 1–6, 2015.

[17] C. E. Scheidegger, H. T. Vo, D. Koop, J. Freire, and C. T. Silva. Querying and re-using workflows with Vstrails. In *SIGMOD*, pages 1251–1254, 2008.

[18] M. Stamatogiannakis, P. Groth, and H. Bos. Looking inside the black-box: capturing data provenance using dynamic instrumentation. In *IPAW*, pages 155–167, 2014.

[19] M. Weske, G. Vossen, and C. B. Medeiros. *Scientific workflow management: WASA architecture and applications*. Citeseer, Universität Münster. Angewandte Mathematik und Informatik, 1996.