Revealing the Detailed Lineage of Script Outputs using Hybrid Provenance

Qian Zhang¹, Yang Cao¹, Qiwen Wang², Duc Vu³, Priyaa Thavasimani⁴, Timothy McPhillips¹, Paolo Missier⁴, Peter Slaughter⁵, Christopher Jones⁵, Matthew B. Jones⁵, Bertram Ludäscher¹, ²

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

github.com/yesworkflow-org/yw-idcc-17

Motivation: **Provenance** is good for self and others ...

- Data- and Workflow-Provenance are crucial for **transparency** and **reproducibility** in computational and data-driven science.
- Scientific workflow systems provide both **prospective provenance** (workflow graphs) and **retrospective provenance** (runtime observables).

http://ajdcreative.com.au/thoughts/a-w-food-packaging-wine-label-design-trend-series#WKsMFRKLSRs
... but there are some Challenges ...

- Most computational analyses and workflows are conducted using *scripts* (Python, R, MATLAB, ...) rather than workflow systems.

- **Retrospective Provenance Observables**
  - e.g. DataONE RunManagers (file-level), ReproZip (OS-level), or noWorkflow (Python code-level)

  ... only **yield isolated fragments** of the overall data lineage and processing history.

- **Prospective Provenance** links and contextualizes fragments into a meaningful and comprehensible workflow, but *scripts alone do not reveal the underlying workflow*.

- Provenance (like other metadata) appears to be **rarely actionable or immediately useful** for those who are expected to provide it (“provenance for others”).
Approach

• Simple YesWorkflow (YW) annotations allow authors to reveal the workflow (prospective provenance) implicit in scripts.

• Prospective provenance queries expose and test data dependencies at the workflow level.

• Hybrid provenance queries situate runtime observables (retrospective provenance) in the overall workflow, yielding meaningful knowledge artifacts.

→ Comprehensible workflow graphs & customizable provenance reports for script runs, along with data and code in scientific studies ("provenance for self").
Hybrid provenance
Prospective element: *script-based workflow graph*

- YesWorkflow (YW) tool
  - [try.yesworkflow.org](http://try.yesworkflow.org)
  - YW models script as workflow with YW annotations
    - Benefits script authors, future code readers, and/or general users
  - **language-independent**: Python, MATLAB, JAVA, R, C/C++, bash
  - **user-declared**:
    - **Author** adds simple YW annotations to reveal the hidden workflow (W)
      - @begin, @end, @in, @end
    - **Model** what’s important
  - Script users can see W’s structure graphically (prospective provenance graph), including its underlying dataflow structure

- **Visualize** or query **prospective provenance** *before* running a script.
Prospective provenance queries

• **Workflow-structure (general) queries**
  - Show/render the complete prospective provenance graph.
  - List all of the code blocks
    - defined in the script along with any description given for each.
    - nested (directly or indirectly) within a particular code block.
    - that invoke a particular function or external program.
    - that contain a particular block (directly or indirectly).
    - that receive inputs derived (directly or indirectly) from the outputs of a particular upstream code block.
    - affected (directly or indirectly) by a particular parameter value provided to the script.

• **Prospective data provenance (general) queries**
  - Reveal the upstream lineage / complete derivation of any final or intermediate data product y, (i.e., the subgraph of data and steps upstream of y in W).
    - List the inputs to the script
    - List the computational steps (code blocks) involved
  - Compute the downstream influence of x (i.e., the subgraph of the data derived from x along with the derivation steps).
    - Given a data item to what other downstream items does it contribute? i.e., List the outputs that depend on / downstream of a particular script input.
  - For a particular computational step reveal where each input to the step comes from: an input to the script, a constant in the script, or a value produced by a different step, for example.
% Algorithm 1: method used in MstMIP
% Examine the type of each pixel to see if it inc
% BEGIN examine_pixels_for_gross
% End Rain @AS Rain Matrix
% Rout C3 @AS C3 Data
% Rout C4 @AS C4 Data
89 C3-ones(ncols, nrows)“*<999.0”;
90 C4-ones(ncols, nrows)“*<999.0”;
91 for j=1:nrows
92 frac_c3=0.0;
93 frac_c4=0.0;
94 if (Grass(i,,j)>0)
95 ngrow=0;
96 mmonth_c3=0;
97 mmonth_c4=0;
98 for m=1:12
99 if (Tair(i,,j)==278)
100 ngrow=ngrow+1;
101 end
102 if (Tair(i,,j)==295)
103 mmonth_c3=mmonth_c3+1;
104 elseif (Tair(i,,j)==295 & Rain(i,,j)=0)
105 mmonth_c4=mmonth_c4+1;
106 elseif (Tair(i,,j)==295 & Rain(i,,j)==1)
107 mmonth_c4=mmonth_c4+1;
108 end
109 end
110 if (mmonth_c3==12)
111 frac_c3=1;
112 frac_c4=mmonth_c4/ngrow;
113 end
114 elseif (mmonth_c4==1)
115 frac_c4=1;
116 frac_c4=mmonth_c4/ngrow;
117 end
118

simple YW annotations

Your formerly hidden workflow REVEALED!
- INPUTS, OUTPUTS, and key STEPS
What `C4_fraction_data` depends on ...

```plaintext
fetch_monthly_mean_precipitation_data
fetch_monthly_mean_air_temperature_data

Rain_Matrix
Tair_Matrix
SYNMAP_land_cover_map_data

examine_pixels_for_grass
fetch_SYNMAP_land_cover_map_variable

C4_Data
lon_variable
lat_variable
lon_bnds_variable
lat_bnds_variable

generate_netcdf_file_for_C4_fraction

C4_fraction_data
```

`C4_fraction_data` lineage very similar to overall workflow graph!
What Grass_fraction_data depends on ...

C4_fraction_data lineage different from overall workflow graph!
- Smaller subgraph
- Depends on 1 of 3 inputs!
Retrospective element

• Runtime observables:
  ➢ input, output, and intermediate data product

• Raw runtime observables come from:
  ➢ YW-recon
    ✓ Not a runtime provenance **recorder**
      ▪ YW reconstructs script run and retrospective provenance using **URI template**
  ➢ YW-recon log files
    ✓ Not a runtime provenance **recorder**
      ▪ User-defined **log files** capture runtime observables at any level of granularity
      ▪ YW reconstructs script run and retrospective provenance using @log associated with @out YW annotations
  ➢ DataONE MATLAB RunManagers (RM)
    ✓ MATLAB runtime recorder
      ▪ Capture provenance at file-level
    ✓ Provenance exporter → run observation fact in Prolog
  ➢ noWorkflow (NW) toolkit:
    ✓ Python runtime provenance recorder
      ▪ Capture provenance at both code-level and file-level
    ✓ Provenance exporter → run observation fact in Prolog
    ✓ Query and visualize provenance
Prospective vs. retrospective provenance

• Prospective provenance
  ➢ *Forward* data provenance describes how a data is used/applied *after* it has been created.
  ➢ Approach:
    ✓ YesWorkflow (YW) tool for modeling scripts as workflows

• Retrospective provenance
  ➢ *Backward* data provenance describes where the data came from and how it was generated *before* it has been created.
  ➢ Approach:
    ✓ Reconstructing provenance with YW-Recon
      o Extra: YW-log
    ✓ Recording provenance with DataONE RunManager (RM)
    ✓ NoWorkflow (NW)
Hybrid provenance

• **Prospective + Retrospective**
  - situate runtime observables (retrospective provenance) in the overall workflow, yielding meaningful knowledge artifacts
  - 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)

• **Hybrid provenance bridge**
  - Logic rules for reconstructing, querying, and visualizing prospective and retrospective provenance together
  - Integrate provenance gathered from file names and paths, log files, data file headers, run metadata, and records of run-time events.
    - YW-RM bridge at the file-level
    - YW-NW bridge
      - At the file-level
      - At the code-level
(General) hybrid queries and views

• Show/render reconstructed provenance graph with all observables in context of the workflow.

• Backward lineage query:
  ➢ Given a file output by a run of a script, indicate the input files from which it was derived or by which it was affected.
  ➢ Show/Render reconstructed workflow graph upstream of a given data product.

• Forward lineage query:
  ➢ Given an input file to a script, indicate which output files were derived from or affected by the data contained in that file.
  ➢ Show/Render reconstructed workflow graph downstream of a given data product.
What happens after running the script?

Hybrid provenance graph!

- 3 inputs spread across 25 (=2x24 + 1) files
- Do all 3 output files depend on all 25 inputs?
What C4_fraction_data depends on (hybrid) ...

Earlier prospective query result
What Grass_fraction_data depends on ...

**Upstream of Grass_fraction_data (hybrid)**

**Overall workflow**

**Upstream of Grass_fraction_data (prospective)**
### Other use cases spanning multiple disciplines

<table>
<thead>
<tr>
<th>Domain</th>
<th>Use case</th>
<th>Programming language</th>
<th>Provenance methods</th>
</tr>
</thead>
<tbody>
<tr>
<td>Climate science</td>
<td>C3C4</td>
<td>MATLAB</td>
<td>YW + MATLAB RunManager</td>
</tr>
<tr>
<td>Astrophysics</td>
<td>LIGO</td>
<td>Python</td>
<td>YW + NW (code-level)</td>
</tr>
<tr>
<td>Protein crystal samples</td>
<td>Simulate data collection</td>
<td>Python</td>
<td>YW + NW (code-level)</td>
</tr>
<tr>
<td>Biodiversity data curation</td>
<td>kurator-SPNHC</td>
<td>Python</td>
<td>YW-recon + YW-logging</td>
</tr>
<tr>
<td>Social network analysis</td>
<td>Twitter</td>
<td>Python</td>
<td>YW + NW (file-level)</td>
</tr>
<tr>
<td>Oceanography</td>
<td>OHIBC Howe Sound (multi-run multi-script)</td>
<td>R</td>
<td>YW + R RunManager</td>
</tr>
</tbody>
</table>
demo site: github.com/yesworkflow-org/yw-idcc-17
LIGO example: What `strain_L1_whitenbp` depends on ...

- Intermediate data `strain_L1_whitenbp` depend only on 2 out of 5 inputs!
- 3 inputs spread across 5 (=2x2+1) files
- Does intermediate data `strain_L1_whitenbp` depend on all 5 inputs?
YW-log implementation for the SPNHC biodiversity data cleaning use case

```plaintext
# YW-log implementation for the SPNHC biodiversity data cleaning use case

## Initialize run
- `Create the run log file`
- `name_cleaning_logFileName: name_val_log.txt`

## Read scientific name
- `Read scientificName from local authority source`
- `local_authority_source_scientificName_list`

## Read data records
- `Read original dataset`
- `record_id_dataFileName: record_id.txt`

## Check if scientificName is nonempty
- `Check if scientificName value is present`
- `empty_scientificName`
- `nonEmpty_scientificName`
- `log_name_is_empty`

## Find name match
- `Find if the scientificName matches any record in the local authority source using exact and fuzzy match`
- `match_result`
- `match_method`
- `matching_record`
- `log_match_not_found`
- `update_scientific_name`
- `updated_scientificName`
- `update_authorship`
- `updated_authorship`
- `log_updated_record`
- `log_accepted_record`

## Write data into a new file
- `Write data into a new file`
- `data_with_cleaned_namesFileName: demo_output_name_val.csv`

## Log summary
- `Summarize on all the records`
- `original_datasetFileName: demo_input.csv`
- `local_authority_sourceFileName: demo_localDB.csv`
```

---

![Diagram of YW-log implementation for the SPNHC biodiversity data cleaning use case](image-url)

---

**Log Example:**

- **2017-01-06-01:15:46** Reading input record 17670878
- **2017-01-06-01:15:46** Checking if scientificName value is empty - MD
- **2017-01-06-01:15:46** Trying external check local authority source EXACT match for validating scientificName: 'Placopten magellanicus'
- **2017-01-06-01:15:46** EXACT match was FAILED, compliant with local authority source: FAILED
- **2017-01-06-01:15:46** Fuzzy external check local authority source FUZZY match for validating scientificName: 'Placopten magellanicus'
- **2017-01-06-01:15:46** FUZZY match was SUCCESSFUL, compliant with local authority source: SUCCESSFUL
- **2017-01-06-01:15:46** UPDATING record 17670878: scientificName from 'Placopten magellanicus' to 'Placopten magellanicus'
- **2017-01-06-01:15:46** ACCEPTED the record 17670878

---

**Log File:**

- `log_variable_value(resource_id, log_entry_id, log_variable_id, log_variable_value)
- log_variable_value(3, 1, '2017-01-06-01:15:46')
- log_variable_value(3, 2, 'demo_input.csv')
- log_variable_value(3, 3, '17670878')
- log_variable_value(3, 4, 'Placopten magellanicus')
- log_variable_value(3, 5, 'SUCCESSFUL')
- log_variable_value(3, 6, 'accepted')`
Conclusions

• **Provenance** from script runs can **be revealed graphically and made actionable** (e.g., to yield customizable data lineage reports) via
  - simple YW user annotations
  - linking runtime observables (e.g. DataONE RunManager, noWorkflow), and
  - sharing provenance artifacts and executable queries.

• Hybrid provenance can be created by
  - prospective provenance + file-level runtime observables
  - prospective provenance + code-level runtime observables

• Hybrid view
  - the dual views of retrospective and prospective provenance can:
    - reveal the overall story of a script run: For documenting and explaining script-based scientific workflows to others
    - enable researchers to query and explore the detailed derivation history of particular data products yielded by script runs
Future work

- Multi-run/script (ongoing)
- Extend YW toolkit to
  - support other (optional) workflow modeling constructs (e.g., control-flow to complement dataflow)
  - to support graph pattern queries
  - to support project-level provenance.
- Evolve ProvONE to support project-level provenance and graph queries
  - to enable mapping between YW and ProvONE data model s.t. ProvONE compatible vocabulary extensions may be used in YW in the future
  - to generate and query ProvONE-compatible RDF representations of YW annotations, workflow models, and retrospective provenance, etc.
Questions?

Thank you!

• Demo site: https://github.com/yesworkflow-org/yw-idcc-17
• **Script** (Python, MATLAB, R, …)

• **Script-based workflow (W) model**
  - $\leftrightarrow$ high-level prospective provenance graph that can be visualized
  - User-declared
  - Extracted via the YW toolkit (try.yesworkflow.org)
  - For script authors / future code readers / general users to have a high-level overview of the W’s structure, including its underlying dataflow structure

• **Query** the database related to the *conceptual workflow model* $\leftrightarrow$ prospective provenance queries
  - to reveal dataflow dependencies in the workflow model

• **Script run**
  - Runtime observables: input, output, and intermediate data product

• **Query** the database regarding the *runtime observables* $\leftrightarrow$ retrospective provenance queries
  - link the *conceptual* workflow entities with observations made about the script run
  - combine the information extracted from a marked-up script with references to data files corresponding to a run of that script

• **Hybrid query:**
  - **Hybrid provenance:** prospective + retrospective
    - situate runtime observables (retrospective provenance) in the overall workflow, yielding meaningful knowledge artifacts
    - 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)
  - $\rightarrow$ Hybrid views, queries, and visualizations
  - the dual views of retrospective and prospective provenance can:
    - reveal the overall story of a script run: $\rightarrow$ For documenting and explaining script-based scientific workflows to others
    - enable researchers to query and explore the detailed derivation history of particular data products yielded by script runs