A Model for
User-Oriented Data Provenance
in Pipelined Scientific Workflows

Timothy McPhillips¹
Shawn Bowers¹
Bertram Ludaescher¹,²
Shirley Cohen³
Susan Davidson³

¹ UC Davis Genome Center
² Department of Computer Science
University of California, Davis
³ Computer and Information Science
University of Pennsylvania

Outline

• Motivation
  – supporting user-oriented provenance queries
  – in the presence of complex (e.g. pipelining) MoCs
• Basic Workflow & Provenance Framework
  – Workflow graph, parameters, inputs
  – Models of Computation (MoCs): PN, SDF, DAG,…
• Pipelined, Collection-Oriented Workflows
• Scientist-oriented provenance queries
• The Read-Write-ReSet (RWS) Model of Provenance
• Conclusions
Provenance for the WF Engineer / Plumber

- A Workflow Engineer’s View
  - Monitor, benchmark, and optimize workflow performance
  - Record resource usage for a workflow execution
  - “Smart Re-run” of (variants of) previous executions
  - Checkpointing & restart (e.g. for crash recovery, load balancing)
  - Debug or troubleshoot a workflow run
  - Explain when, where, why a workflow crashed

Provenance for Domain Scientists!

- Query the lineage of a data product
  - from what data was this computed? (“real” dependencies please!)
- Evaluate the results of a workflow
  - do I like how this result was computed?
- Reuse data products of one workflow run in another
  - (re-)attach prior data products to a new workflow
- Archive scientific results in a repository
- Replicate the results reported by another researcher
- Discover all results derived from a given dataset
  - … i.e. across all runs
- Explain unexpected results
  - … via parameter-, dataset-, object-dependencies in the scientist’s terms (yes, you may think “ontology” here … )
Example Scientific Workflow (Kepler)

Basic Notions

- Workflow graph $W$ (actors, ports, channels)
- Parameters $P$
- Data input $i$, output $o$

Given $W$, $P$, and $i$, is $o$ determined??

- No! $W(P, i) =$?

- Missing Model of Computation (Moc) $M$:
  - Now: $o = M(W, P, i)$
  - Example MoCs: $M = \{DAG, SDF, PN, \ldots, DE, CT, \ldots\}$
Models of Computation (MoCs)

- **PN (Process Network)**
  - actors are independent processes/threads
  - buffered, unidirectional channels
  - pipelined, asynchronous execution
  - unbounded buffers (e.g. fast producers, slow consumers)
  - loops OK (but beware of deadlocks)

- **SDF (Synchronous Data-Flow)**
  - actors declare token consumption & production rates
  - static “firing” schedule (pos. integer solutions to balance equations)
  - fixed buffer size
  - loops OK (deadlocks statically detectable)

- **DAG (Directed Acyclic Graph)**
  - limited, special case of SDF (token rate is 1-in, 1-out)
  - each actor fires exactly once (topological sort yields schedule)
  - NO loops

Observables

- **Model of Computation MoC M**
  - specification/algorithm to compute o = M(W,P,i)
  - a director or scheduler implements M
  - gives rise to formal notions of
    - computation (aka run) R; typically tree models

- **Model of Provenance MoP M’**
  - approximation M’ of M
  - a trace T approximates a run R by inclusion/exclusion of observables
  - “ T = R − ignored-observables + non-functional-observables ”

- **Observables (of a MoC M)**
  - functional observables (may influence output o)
    - token rate, notions of firing, …
  - non-functional observables (not part of M, do not influence o)
    - token timestamp, size, … (unless the MoC cares about those)

- What is a good model of provenance? What is a good provenance schema?
**Observables for SDF**

<table>
<thead>
<tr>
<th>SDF</th>
<th>statically determinable</th>
<th>@runtime</th>
<th>functional (part of MoC)</th>
<th>non-functional (not part of MoC)</th>
</tr>
</thead>
<tbody>
<tr>
<td>token production, consumption</td>
<td>✓</td>
<td></td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>firing</td>
<td></td>
<td>✓</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>firing order</td>
<td>✓</td>
<td></td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>timestamp, token size, ...</td>
<td>✓</td>
<td></td>
<td>✓</td>
<td></td>
</tr>
</tbody>
</table>

**Provenance for Domain Scientists!**

- **Query the lineage of a data product**
  - from what data was this computed? ("real" dependencies please!)
- **Evaluate the results of a workflow**
  - do I like how this result was computed?
- **Reuse data products of one workflow run in another**
  - (re-)attach prior data products to a new workflow
- **Archive scientific results in a repository**
- **Replicate the results reported by another researcher**
- **Discover all results derived from a given dataset**
  - ... i.e. across all runs
- **Explain unexpected results**
  - ... via parameter-, dataset-, object-dependencies in the scientist's terms (yes, you may think "ontology" here ... )
**Example:** Inferring phylogenetic trees by comparing DNA sequences from different organisms

```
AGGGCAT  TAGCCCA  TAGACTT  TGCACAA  TGCCTTT
```

![Tree diagram](http://www.phylo.org/)

**Steps in Phylogenetic Analysis**

1. **Gather** data.
2. **Align** sequences.
3. **Reconstruct phylogeny** on the multiple alignment - often obtaining a large number of trees ("nested collections")!
4. **Compute consensus** (or otherwise estimate the reliable components of the evolutionary history).
5. Perform **post-tree analyses**.
Scientific provenance questions we can ask about a run of this workflow

- What DNA sequences were input to the workflow (this run)?
- What phylogenetic trees were output by the workflow?
- What phylogenetic trees were created (intermediate or final) by the workflow?
- What actor created this phylogenetic tree?
- What sequences input to the workflow does this consensus tree depend on?
- What input sequences were not used to derive any output consensus trees?
- What was the sequence alignment (key intermediate data) used in the process of inferring this tree?
- Which actors were involved in creating this tree?
Snapshot of a pipelined workflow operating on independent sets of sequences

Pipelined Collection-oriented Workflows

Metada for collection b

Actors 4 and 5 are processin contents of collection a concurrently

Acting 4 processes entire collections (of a particular type) at one time.

Acting 5 processes one data token at a time.
Scientific provenance questions we can ask about a run of this workflow

- What DNA sequences were input to the workflow (this run)?
- What phylogenetic trees were output by the workflow?
- What phylogenetic trees were created (intermediate or final) by the workflow?
- What actor created this phylogenetic tree?
- What sequences input to the workflow does this consensus tree depend on?
- What input sequences were not used to derive any output consensus trees?
- What was the sequence alignment (key intermediate data) used in the process of inferring this tree?
- Which actors were involved in creating this tree?

How can we answer these kinds of user-oriented questions for pipelined workflows?

- Need to record what data was available to an actor when an output token was written.
- Recording timestamps for each token-write event is not sufficient: actors may operate on independent sets of data in a single run.
  - actors may reset their state, “forgetting” previously read tokens
- Thus, need to record all token-read (r), token-write (w), and state-reset (s) events for all actors in a workflow run.
  - RWS model of provenance
Read-Write-ReSet Model

- $r, r, \ldots, r, w, w, \ldots, w, r, \ldots, r, w, \ldots, w, \ldots$
- **firing**
- what about actor state? what about “real” dependencies?
- reset event $s$ defines when actor “cuts off” dependencies
  - a semantic notion, known to the actor [developer] (or part of a higher-order scheme)
- $r, r, \ldots, r, w, w, \ldots, w, [s!] r, \ldots, r, w, \ldots, w, \ldots$

RWS event log for a run of the workflow

<table>
<thead>
<tr>
<th>$E_{wcc}$</th>
<th>$E_{rwp}$</th>
<th>$E_{wch}$</th>
<th>$E_{ref}$</th>
<th>$E_{wcc}$</th>
<th>$E_{rwp}$</th>
<th>$E_{wch}$</th>
<th>$E_{ref}$</th>
<th>$E_{wcc}$</th>
<th>$E_{rwp}$</th>
<th>$E_{wch}$</th>
<th>$E_{ref}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$p_0$</td>
<td>$w t_9$</td>
<td>1</td>
<td>$A_1$</td>
<td>$s$ - 3</td>
<td>$A_2$</td>
<td>$s$ - 4</td>
<td>$p_3$</td>
<td>$r t_{55}$</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$\ldots$</td>
<td></td>
<td></td>
<td>$\ldots$</td>
<td></td>
<td>$\ldots$</td>
<td></td>
<td>$\ldots$</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$p_0$</td>
<td>$w t_{10}$</td>
<td>1</td>
<td>$p_1$</td>
<td>$r t_{99}$</td>
<td>1</td>
<td>$p_3$</td>
<td>$w t_{99}$</td>
<td>1</td>
<td>$p_3$</td>
<td>$r t_{55}$</td>
<td>1</td>
</tr>
<tr>
<td>$A_1$</td>
<td>$s$ - 1</td>
<td></td>
<td>$A_3$</td>
<td>$s$ - 1</td>
<td>$p_3$</td>
<td>$w t_{99}$</td>
<td>1</td>
<td>$A_4$</td>
<td>$s$ - 2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$p_1$</td>
<td>$r t_{9}$</td>
<td>1</td>
<td>$p_3$</td>
<td>$w t_{31}$</td>
<td>3</td>
<td>$p_3$</td>
<td>$w t_{99}$</td>
<td>1</td>
<td>$p_3$</td>
<td>$r t_{55}$</td>
<td>2</td>
</tr>
<tr>
<td>$\ldots$</td>
<td></td>
<td></td>
<td>$\ldots$</td>
<td></td>
<td>$\ldots$</td>
<td></td>
<td>$\ldots$</td>
<td></td>
<td>$\ldots$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$p_1$</td>
<td>$r t_{9}$</td>
<td>1</td>
<td>$p_3$</td>
<td>$w t_{31}$</td>
<td>3</td>
<td>$p_3$</td>
<td>$w t_{99}$</td>
<td>1</td>
<td>$p_3$</td>
<td>$r t_{55}$</td>
<td>2</td>
</tr>
<tr>
<td>$A_1$</td>
<td>$s$ - 2</td>
<td></td>
<td>$A_3$</td>
<td>$s$ - 2</td>
<td>$p_3$</td>
<td>$w t_{99}$</td>
<td>1</td>
<td>$A_4$</td>
<td>$s$ - 3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$p_1$</td>
<td>$r t_{9}$</td>
<td>2</td>
<td>$p_3$</td>
<td>$r t_{50}$</td>
<td>2</td>
<td>$p_3$</td>
<td>$w t_{99}$</td>
<td>2</td>
<td>$p_3$</td>
<td>$r t_{55}$</td>
<td>1</td>
</tr>
<tr>
<td>$\ldots$</td>
<td></td>
<td></td>
<td>$\ldots$</td>
<td></td>
<td>$\ldots$</td>
<td></td>
<td>$\ldots$</td>
<td></td>
<td>$\ldots$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$p_1$</td>
<td>$r t_{56}$</td>
<td>2</td>
<td>$A_2$</td>
<td>$s$ - 3</td>
<td>$A_4$</td>
<td>$s$ - 1</td>
<td>$p_7$</td>
<td>$r t_{64}$</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$p_2$</td>
<td>$w t_{20}$</td>
<td>2</td>
<td>$p_3$</td>
<td>$r t_{21}$</td>
<td>3</td>
<td>$p_7$</td>
<td>$r t_{64}$</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Scientists’ questions to queries against the execution log

- What DNA sequences were input to the workflow? This is one of the first questions a scientist might ask about the workflow run. Given an object type $\text{seq}$, the parameterized query

$$q_1(\text{seq}) := \{ o \mid t \in \text{token}(T) \land \text{writer}(t) \in \text{in}(W) \land \text{object}(t) = o \land o \in \text{type}(\text{seq}) \}.$$ 

returns the set of objects of type $\text{seq}$ that were input to the workflow run. For our example trace, $q_1(\text{Sequence})$ returns the objects $\text{seq}_1$ to $\text{seq}_{10}$. The expression $t \in \text{token}(T)$ selects a token from the trace, the expression $\text{writer}(t) \in \text{in}(W)$ checks that the token was written by an input port of the workflow $W$, the expression $\text{object}(t) = o$ obtains the object associated with $t$, and the expression $o \in \text{type}(\text{seq})$ verifies that $o$ has $\text{seq}$ as a type.

- What phylogenetic trees were output by the workflow? This is another basic question that a scientist might initially ask after a run. Given the query

$$q_2(\text{seq}) := \{ o \mid t \in \text{token}(T) \land \text{reader}(t) \in \text{out}(W) \land \text{object}(t) = o \land o \in \text{type}(\text{tree}) \},$$

the expression $q_2(\text{Tree})$ returns the objects $\text{tree}_1$ and $\text{tree}_2$.

- What phylogenetic trees (intermediate or final) were created by the workflow? This question requests both intermediate as well as final data products of a run. Given the query

$$q_3(\text{seq}) := \{ o \mid t \in \text{token}(T) \land \text{writer}(t) \notin \text{in}(W) \land \text{object}(t) = o \land o \in \text{type}(\text{seq}) \}.$$ 

**Queries could also be defined in Datalog or in query languages for graphs or semi-structured data.**

Dependency graphs for the workflow run

- **Token dependency graph**
- **Object dependency graph**
- **Invocation dependency graph**

These graphs can be constructed easily (e.g. as Datalog queries) from the event log.
3.4 Dependency Graphs

Using the RWS model, we are able to infer from the event log the token dependency graph. That is, for each token \( t \), we can know which parent tokens \( \{t_1, \ldots, t_k\} \) directly contributed to the production of \( t \) (as the result of an actor firing). As an example, in the upper left of Fig. 1, \( \{t_1, \ldots, t_7\} \) are parent tokens of \( t_9 \). Conversely, \( t_9 \) is the parent of \( t_{12}, t_{25}, t_{26} \). The following Datalog program illustrates how the token dependency graph can be computed from the event log.

The event relation corresponds to the event log and the actor relation contains a mapping from ports to their corresponding actors.

\[
\text{depends-on}(T_1, T_2) : - \text{event}(P_1, a, T_1, C_1), \text{event}(P_2, r, T_2, C_2), \\
\text{actor}(P_1, A), \text{actor}(P_2, A), \text{reset}(A, C_3, C_4), \\
C_k \leq C_2 \leq C_1 < C_n, \\
\text{reset-between}(A, C_k, C_1), \\
\text{event}(A, a, \ldots, C_1), \text{event}(A, a, \ldots, C_n), \\
\text{event}(A, \ldots, X, C), \text{event}(A, \ldots, C_n).
\]

We say that \( T_1 \) depends on \( T_2 \) whenever \( \text{depends-on}(T_1, T_2) \) is true.

In addition to the token dependency graph, we are also able to infer the object dependency graph using the RWS model. Object dependencies describe user data.

### Running-Average actor

- Actor A1 computes a sequence of running temperature averages from a series of input temperature readings. Each average reading is dependent on all temperature readings received since the most recent state-reset of the actor.
- Note that assuming that an implicit state-reset follows each write event would be incorrect, because this would imply that each temperature average depended only on the latest temperature reading received. The state reset event must be recorded explicitly.
Filtering actor

- Actor A3 consumes a series of protein structures and outputs only those structures meeting a minimum resolution. All other input protein structures are **discarded**. Because the state-reset events are recorded, it is clear that output token $fs3$ **depends only** on input $s3$, and not on $s2$, even though no write event separates the read events for $s2$ and $s3$.

Iterative execution

- Actor A3 requires a random number seed to initiate a search for phylogenetic trees. It is fired iteratively for a particular character matrix $cm$, using a distinct seed $se$ on each iteration. Recording state reset events allows each tree inferred in this way to be associated not only with the character matrix from which it was derived, but also with the particular random number seed used.
**Sliding window**

- Actor A5 operates on a sliding window of data, predicting the secondary structure of a protein, residue by residue, based on the types of residues within a contiguous segment of the protein chain. In this case, introducing an explicit forget (f) event would be more efficient.

**Conclusions**

- There are more complex computation models than DAG
  - pipeline parallelism (cf. streaming data, continuous queries)
  - nested data (collection-oriented workflows)
  - nested workflow definitions
  - require some thoughts about the provenance schema PS

- Plumbing is useful and needed
  - … but “scientific” (conceptual) queries of execution logs may be what scientists really want

- Towards User-driven Provenance Schema Design
  - Given sets of user queries \(?- U\) , desired MoCs \{DAG, SDF, PN, ..\}
  - Problem: Design a provenance schema PS such that queries U can be answered!
  - Solution: …
Acknowledgements

Scientific data management, data integration, scientific workflows, workflow provenance, collaboration with domain scientists

- Bertram Ludaescher
- Shawn Bowers
- Tim McPhillips

Ullas Nambar
Norbert Podhorski
Dave Thau
Tim Wong
Alex Chen

Kepler members
Ikay Altintas
Chad Berkley
Shawn Bowers
Matthew Brooke
Christopher Brooks
Zhengang Cheng
Tobin Fricke
Daniel Higgins
Efrat Jaeger
Matt Jones
Werner Krebs
Edward A. Lee
Bertram Ludaescher
Timothy McPhillips
Steve Mock
Stephen Neuendorffer
Rod Spears
Wibke Sudholt
Jing Tao
Mladen Vouk
Xiaowen Xin
Yang Zhao

User-Oriented Data Provenance, McPhillips et al.