Provenance and the Price of Identity

Adriane Chapman and H.V. Jagadish

University of Michigan
Ann Arbor, MI USA
{apchapma, jag}@umich.edu

Abstract. As developers acknowledge that provenance is essential, more and more datasets are attempting to keep provenance records describing how they were created. Some of these datasets are constructed using workflows, others cobble together processes and applications to manipulate the data. While the provenance needs are the same, the inputs and set of processes used must be kept, the identity needs are very different. We outline several identification strategies that can be used for data manipulation outside of workflows. We evaluate these strategies in terms of time to create and store identity, and the space needed to keep this information. Additionally, we discuss the strengths and weaknesses of each strategy.

1 Introduction

Workflow systems [1, 2, 4, 10, 17, 19, 24–26] provide a framework for users to map out a set of inputs and a series of processes to manipulate that input. In the First Provenance Challenges [19], the given inputs were a set of brain images, and a set of reference brain images. A series of processes such as "align_warp" and "softmean" were arranged in a particular order to output a set of graphics. As the workflow runs on a set of data, provenance information captures the details of what happened to that data. In most of these systems, particularly the ones which utilize myGRID [11, 12, 18, 26], the notion of data identity is firmly established and rigorously attended to. For example, in the Taverna workflow system [26], every computed data product is given a Taverna identity. This strict notion of identity in provenance is essential for data reuse and distribution of resources.

However, as the world at large achieves a greater understanding for the need of maintaining provenance, more and more hand-built systems are attempting to store provenance. These systems and datasets are built outside traditional workflow frameworks. Many times, these systems and datasets are a series of processes applied in a particular order to a set of inputs, exactly as a workflow would, but without the rigid, and helpful but occasionally overwhelming, workflow framework. In other words, an "implicit" workflow is followed to create a desired outcome. Some examples of these, hand-made systems are:

– The creation of MiMI [15]: A set of classic protein interaction sources are transformed, merged and annotated into one cohesive dataset.

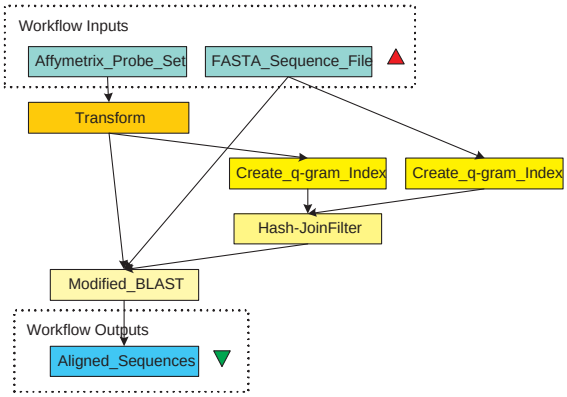


Fig. 1. The MANIPULATIONs used by miBLAST [16] to align probe and DNA sequences.

– The creation of the Linguist Search Engine [23]: Sentences are culled from the web. These sentences are then run through a series of parsing, tokenization and part of speech tagging processes. The end result is a database of sentences that linguists can query using sentence syntax.

A more in-depth example of a system, and how it strings together many processes workflow-style, is explored below.

Example 1 *miBLAST [16] finds sequence alignments between 25bp Affymetrix Probe sets and published gene sequences, and is 10x faster than a traditional BLAST search. Figure 1 shows a simplified breakdown of the processes used. First, probe sequence data from an Affymetrix probe is transformed into a specific format. Then, a q-gram index with specified word size is then built on the Query Set of Affymetrix probes. A q-gram index, with specified word size, is also built for the Database Set of sequences. These two indexes are then input into a Hash-Join Filter Module that outputs a set of (Query Set, Database Set) pairs based on word overlap. This set of paired sequences are then fed into a modified BLAST. Instead of doing a sequence comparison between every Query Sequence and every Database Sequence, only sequence comparisons between sequences in a (Query Set, Database Set) pair occur.*

We would like to re-iterate that while the drawing in Figure 1 is the same as is used by many workflow systems, miBLAST is not created using a traditional workflow. Instead a human user, with a clear goal in mind, runs a set of data through a series of processes.

Many of the systems used to create a dataset are only interested in the final product, and not on any intermediate data produced. If these systems wish to record provenance information, what do they need to do? Obviously, there needs to be some storage system, and some way, even if it is hand-entry, to capture the provenance information [13]. A trickier topic that needs to be addressed by each provenance-capturer centers around identity. How should they identify data products or intermediates, so that the final provenance store can accurately trace what happened to a data item?

As stated earlier, traditional workflow systems have a rigid, fail-safe method of making sure that all intermediate objects can be identified and located. Thus the final version can easily trace through a series of manipulations and intermediate data products if necessary. However, this notion of identity may be overkill for these smaller hand-made systems that do not care about intermediate data reuse or resource distribution. In fact, the protocols performed by many of the traditional workflow systems may add too much overhead to be 'worth it' for a smaller system dedicated to producing one particular dataset.

At some level, there is no escaping the identification problem. For provenance to be useful, there must be some way to uniquely identify objects, and trace their changes. There do not exist universal identifiers for most objects. Even in the few that are uniquely identified, such as protein sequence by RefSeq [22], there are still problems with overlap of equivalent objects with different identifiers, or identifiers being deprecated. However, just because we must uniquely identify objects used does not mean that we are limited to the strategies employed by traditional workflow systems, as long as the provenance attached to a data object clearly and precisely determines the object's history. In the miBLAST example shown in Figure 1, since the output of the Transform process is used multiple times, it is a good idea to uniquely identify all data items produced. On the other hand, it makes less sense for every index entry, the data product of Create_q-gram_Index, to be uniquely identified; the index is only used once, and is easily re-created. Moreover, the provenance attached to each data item in the index will uniquely identify the original sequence used to create it, thus clearly and precisely determining the data item's history.

In this work, we examine the costs and benefits that different identity strategies can afford. We lay out the few assumptions and definitions we use in the rest of this work in Section 2. Section 3 documents and explains the methods currently used when dealing with data and identity. In Section 4, we evaluate these methods. Finally, we discuss implications, related work and conclusions in Sections 5–7.

2 Foundations

Throughout this work, we call the basic logical data unit a *data item*. Data items may be tuples in a relational table, elements in XML, objects of arbitrary granularity in an OODB, etc. One data item may completely include, overlap with, or be totally disjoint from another data item. A data item contains a set of *features*. A data item that is a tuple contains features that are attributes; a data item that is an XML element contains features that are child elements or attributes. Each feature is associated with a data value. Features can be single or multi-valued. A *dataset* is comprised of a set of data items.

We restrict our discussion of identity to the provenance found within systems that perform a series of modifications upon a set of data; we do not look at annotation [3, 27] or lineage [21] system needs. Moreover, we wish to be more general than standard explicit workflow systems [1, 2, 4, 10, 17, 19, 24–26], and also include systems that are built by "implicit" workflows: workflows that refer to a series of steps executed by a user with a specific goal in mind, but without using a workflow system. For example,

a series of relational database queries, or a set of batch files run over some data can be considered an implicit workflow. Indeed, while drawn using a standard workflow representation, miBLAST in Figure 1 is actually an implicit workflow, with the series of processes held together by some hand-written wrapping code. For both explicit and implicit workflow systems, we assume there is an input, a manipulation on that input, and an output. The output may then be used as input to another manipulation. Additionally, we look at the effects of identity on fine-grained provenance, i.e. provenance that can be attached at the dataset, data item or feature level.

A MANIPULATION is a basic unit of processing in a workflow, explicit or implicit. Each MANIPULATION takes one or more datasets as input and produces a dataset as output. We write $M(D^{I_1}, D^{I_2}, ...) = D^O$ to indicate that MANIPULATION M takes datasets D^{I_1}, D^{I_2}, etc as input to generate data set D^O as output.

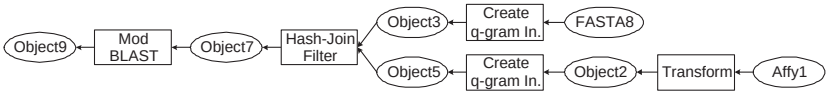
In short, a MANIPULATION is a discrete component of a workflow, and uses a set of specific features from the input dataset. Processes can be mapped into MANIPULATIONs. For example, a MANIPULATION from the Provenance Challenge [19] is:

Manipulation 1 $Softmean$
Takes a set of re-sliced images and headers, and produces a single image and header using the average of the files contents.

An instance of a MANIPULATION applied to a specific data item we call a *manipulation*. We write $m(d^{I_1}, d^{I_2}, ...) = d^O$, where $d^{I_1} \in D^{I_1}$, $d^O \in D^O$, etc. m is an instance of M applied to specific data items d^{I_x} within dataset D^{I_x}. For example, an instance of Create_q-gram_Index with a word size of four is applied to the probe probe1 results in an index entry of {atgc, probe1}.

While a data item is the object in the final result set after all MANIPULATIONs have been performed, we will utilize the following terms: a *intermediate* data item is a data item that is created by a MANIPULATION and utilized by another MANIPULATION(s); an *input* data item is the initial input from the base data into any MANIPULATION; the set of *intermediate* and *input* data items is collectively referred to as *involved* data items.

Regardless of how the information is actually stored, the provenance records we are dealing with contain the following information (for a data item in our running example):



(1)

This can be stored as a series of RDF triplets, a relational table, etc. The important point is that there is a record of the MANIPULATIONs that produced a data item, and it is possible to trace back to the input data item(s).

Finally, we would like to separate the concerns of "storage" and "identity", although they are intricately entwined. "Storage" denotes that a data item has been saved and stored for possible reuse. "Identity", in this work, means that a data item has been uniquely characterized and is immutable. If the data item then changes, a new identity must be assigned. If a data item is stored, it must also have an identity in order to retrieve it. However, data items not stored may also be identified for ease of provenance notation.

3 Current Available Strategies

Within this work, we aim to explore a broad range of techniques to identify input and intermediate data items within a provenance store. We believe there is no one universal 'correct' way to tackle the problem of identifying involved data items. In an attempt to highlight when and where each strategy should and should not be used, we look at four very general classes of involved data item identification and any special algorithms they may need.

3.1 Strong Identification

In a very coarse-grained approach, provenance is only associated at the file-level. Consider a possible identification strategy that will work across many systems: machine_name + path_name. Since every file by definition has a unique path name on a unique machine, this strategy will uniquely provide identity for coarse-grained objects.

In many workflow systems such as [12, 26] that are run with myGRID, every involved data item, not just file, is given a unique identifier. For example, in [28] an LSID of the form URI:urn:lsid:mygrid.ac.uk:data:49841:1 is assigned to every data product, and describes: a prefix, an authority, the authority-specific data namespace, the object identifier and the object version. This LSID is assigned by an authority who maintains responsibility for ensuring that the data item is immutable. For instance, for the workflow in Figure 1 a unique id would be generated and stored with the data product for all data products from transform, Create_q-gram_Index, etc. This level of data product identification is necessary within a workflow context, where each MANIPULATION does not pass data to the next MANIPULATION , but instead writes out data items, which the next MANIPULATION is directed by the workflow system to take as input. Obviously, it is possible to create a provenance record that both associates all MANIPULATIONs with a data item, and can trace back to the source input data items.

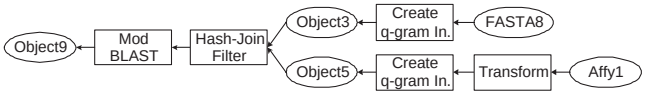
3.2 Strong Identification with IDSet

In [28], the use of an IDSet identifier is discussed to facilitate the use of sets and aggregation. In this system, data items are assigned an LSID as usual. Whenever an aggregation step occurs, the output set of that aggregation is tagged with an ID containing references to all of the ids of the component data items. In our running example, every data product from every component would be identified and stored as discussed above. The only difference would be for the data product of the Hash-Join Filter Module which is an aggregation of entries from the Query Set and the Database Set. The id strategy in this case would be a unique id for the (Q1, D1) pair, that would also contain the unique identifiers of Q1 and D1. Again, because all intermediate data items are being IDed and stored, it is possible to associate all MANIPULATIONs with a data item, and trace back to the source input data items.

3.3 Intermittent Identification

Some results of MANIPULATIONs are only ever used by a unique subsequent MANIPULATION. For example, the results of the Create_q-gram_Index component are only

every used by the Hash-Join Filter component. In this case, it may not be necessary to identify and store every intermediate result. We can effectively store:



(2)

Of course, this does mean that if an intermediate result, $Object2$, is not id-ed and stored, and is later needed by a different MANIPULATION, either the intermediate result must be re-computed, or the MANIPULATION is out of luck. However, to preserve the ability to trace back to input data items, we must be aware of certain restrictions.

Intervals between Identification If there is a long line of MANIPULATIONs that take in no other input but the output of a single previous MANIPULATION, then it may be beneficial to store occasional, intermediate results. For every intermediate result stored, there exists a closer point from which non-stored data items may be recomputed.

Aggregations In [8], it is shown that in certain cases, some intermediate results must be identified and stored for Aggregation results, if the provenance of a particular data item is to be traced back through the aggregation step in ASPJ queries. Moreover, as discussed later, if only intermittent data items are IDed and stored, if a MANIPULATION later needs the results of a previous MANIPULATION whose data items have not been stored, that MANIPULATION (and any MANIPULATIONs prior to it that have not also been IDed and stored) must be re-run. Because an aggregation MANIPULATION can take in data items that may have been produced by a series of MANIPULATIONs , such as HashJoin Filter in Figure 1, the costs of re-running can be explosive. As such, a good rule of thumb is to identify and store data items before use in an aggregation.

Workflows and Intelligent Users If a workflow exists, then a simple traversal of the workflow before running will determine all intermediate data items that must be identified and stored to be used in a subsequent MANIPULATION. For instance in our example, it is readily apparent that the output of the transform component will be used not only for Create_q-gram_Index but also the Modified_BLAST MANIPULATION. These data items should be identified and stored. If a workflow does not exist, all is not lost. An organized user who has a clear goal in mind is almost as good as a workflow, and is likely to know when a particular intermediate output will be reused. Unfortunately, many users are disorganized and run scripts in a haphazard manner. Even worse, often users who wish for intermediate data products are different from those who produced the data. In these cases, intermittent identification may not be the best strategy.

3.4 Initial Identification

An even more extreme way to reduce the number of identifications made would be to label just the input data items. All subsequent intermediate data items are never stored or identified. For example, merely uniquely identifying the input probe and database sequences. The same problem as in Intermediate Identification occurs, in which any intermediate data item that is reused must be recomputed, as stated in re-running below.

3.5 Recreating Intermediate Data Items

The re-creation of intermediate results discussed below requires that MANIPULATIONs be deterministic. Both Intermittent and Input Identification strategies may require intermediate re-creation.

Reverse Transformations Using reverse transformations we can work backwards from a known, identified and stored data item to a prior non-identified or stored intermediate result. [8, 9] show that certain types of MANIPULATIONs can be traced backwards. In particular Select-Project-Join queries require no intermediate results, while Aggregate-Select-Project-Join (ASPJ) queries require selective intermediate results to be stored. For more complicated MANIPULATIONs outside the realm of standard relational operators, a reverse transformation must be explicitly included by the user. Unfortunately, since users rarely take the trouble to define a reverse transformation, we exclude the possibility of using reverse transformations in our evaluation of strategies.

Re-running As long as the input data items and MANIPULATIONs used are stored, the MANIPULATIONs are deterministic and not dependent upon non-repeatable events (such as occurring exactly at noon PST on Valentine's Day 1956), then it is possible to re-run all MANIPULATIONs until the needed intermediate data items are produced. If there are intermediate data items that have been identified and stored, it is possible to re-run using these checkpointed data items as a starting point. This option should be carefully thought through before application, however. Consider a SUM over all book prices in a database. If the database is updated prior to re-running, the re-created intermediate results will be incorrect. In other words, re-running is only an option if the MANIPULATION is deterministic and the data either a) is unchangable, or b) is checkpointed at a previous, re-runnable step.

4 Evaluation

To illustrate the effects of data product identification and storage, we took a real workflow from [14] and, as shown in Figure 2, abstracted each MANIPULATION as: a) an aggregator (aggregate); b) producing data items used only by 1 other MANIPULATION (manipulation); c) producing data items used by multiple MANIPULATIONs (reuse). To facilitate visualizing just how much identity choices matter, we have set up every manipulation in the workflow to output a constant number of data items. For every identification strategy, we run the workflow three times, changing the number of data items used and output between 100, 1,000 and 10,000. Figure 3 contains the set of identity strategies we explored.

Each manipulation was created as a black box with a fixed constant time. Because we only wanted to test the identity component inherent in data manipulation, the manipulations themselves do nothing but sleep for the requisite amount of time and output a new set of data items. An input set of data items is created and fed through the workflow in Figure 2. For Strong ID (S), the data products are sent to an external authority after each manipulation. For Strong ID + IDSet (SS), the same practice occurs, but after

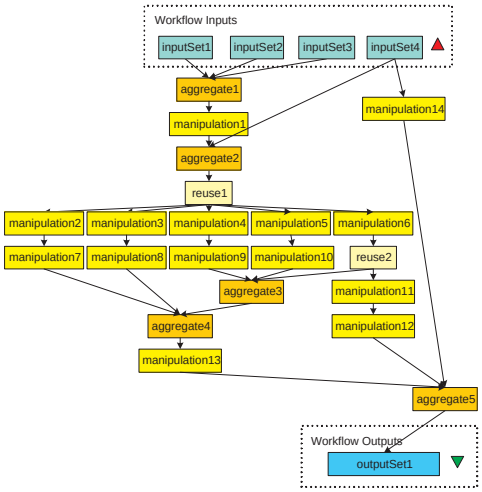


Fig. 2. The series and type of MANIPULATIONs used in the experiments. This workflow is a abstraction of the one found in [14].

S	Strong ID
SS	Strong ID with IDSet
IW	Intermittent ID Storage with prior Workflow knowledge
IAI(2)	Intermittent ID Storage with pre-Aggregation ID Storage and Interval ID Storage every 2 MANIPULATIONS
IAI(3)	Intermittent ID Storage with pre-Aggregation ID Storage and Interval Storage every 3 MANIPULATIONS
IA	Intermittent with Aggregation
I	Input Stored Only

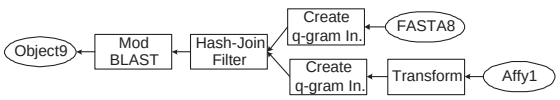
Fig. 3. The strategies employed in all experiments, and their reference codes in the Figures.

every aggregation step, we also build a unique id for the aggregated data item containing all of the input data item ids. Then, we test a series of Intermittent identification strategies. The first of these, Intermittent ID Storage with a known workflow (IW), acts as if a complete workflow is known in advance, and thus any data items that are used in an aggregation, or will be used in multiple MANIPULATIONs are identified and stored. The next three strategies use Intermittent ID Storage, but without advance workflow knowledge. IAI(2) and IAI(3) store object identification before every aggregation MANIPULATION and after every two or three MANIPULATIONs respectively. Finally, Input only ID storage (I) only identifies and stores the input objects.

All experiments were run on a Dell workstation with Pentium 4 CPU at 2GHz with 640MB RAM and 74.4GB disk space with Windows XP. The algorithms were implemented as a Java application. The external object authority was implemented using a mySQL database.

4.1 Pros and Cons

First, we would like to explore some of the pros and cons of each approach in a little more detail. Table 1 contains a breakdown of how each strategy can handle specific situations. With the proper provenance information, all strategies are able to trace back to the input data items. If the provenance structure contains:



(3)

Table 1. The abilities of each general identity strategy.

	Strong	Strong & IDSet	Intermittent	Input Only
Can trace back to original source data	Yes	Yes	Yes	Yes
Intermediate Data Items can be reused by other applications	Yes	Yes	Maybe	No
Processes can be distributed across machines	Yes	Yes	Maybe	No
Lightweight Implementation	No	No	Yes	Yes

In other words, for every data item, there must be an explicit statement about the input data items, even if records of intermediate data items are not maintained. This is a very different approach than the one found in [8, 9] in which the input data items are computed, not stored.

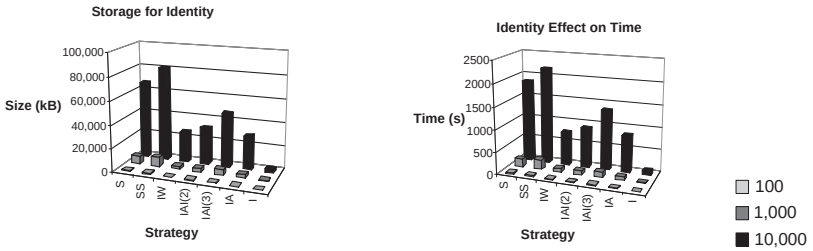
The Strong identification strategies do have a major leg-up over the other techniques: sharing. Because every intermediate data item is stored, these data items can be reused by other MANIPULATIONs within an implicit workflow and other applications. Moreover, because every data item is uniquely identified, it is easier to distribute running the various MANIPULATIONs of an implicit workflow over many different machines.

However, the Intermediate and Input identification strategies do have one significant strength over the Strong approaches: ease of implementation. Consider how many of the datasets created via implicit workflows are actually made. A dataset is ingested, and process1 is applied to it. The results are written to a file. That file is then read in by process2 and new results are written to a file. etc. This is essentially the Input Only Identification strategy. While the data items are stored between MANIPULATIONs in this scenario, they are not identified. Changing this habit and forcing developers to constantly call an external authority for identification is an uphill battle. However, adding an identification step at a few points, such as ingestion, is an easy and lightweight step to take.

4.2 Time and Space

Figure 4 shows the effects of choice of identification strategy on provenance capture time and storage space. The difference between Strong techniques (S), (SS) and Intermittent techniques, (IW), (IAI(2)), (IAI(3)), (IA), is expected since more effort is needed to log every data item produced for the Strong techniques. Unexpected, however, is the size of the difference. For the workflow in Figure 2, utilizing strong techniques is almost twice as expensive in time and space as a basic Intermittent approach. Thus for users and applications that do not require reuse of intermediate data, Intermediate and Input identification strategies are extremely attractive.

Additionally, we would like to point out that the choice of Intermediate identification strategy is important, and is dependent upon the implicit workflow. In Figure 4, we show two Intermediate identification options that store data item identity after set intervals, IAI(2) and IAI(3). In theory, IAI(3) should be faster and smaller to use than IAI(2). However, given the workflow in Figure 2 with which these strategies were utilized, the opposite is the case in both time and space. A quick glance at the workflow



(a) (b) (c)

Fig. 4. The effect of Identity Strategy on space and time while running the workflow found in Figure 2. (a) The size of the provenance store for each strategy. (b) How long each identification strategy takes. (c) Legend for the size of the input set. i.e. every MANIPULATION in Figure 2 takes in 100, 1,000, or 10,000 objects.

shows an Aggregate MANIPULATION almost every two MANIPULATIONs . Thus, for this workflow, IAI(2) is almost equivalent to IA. Unfortunately, because IAI(3) blindly stores results every three MANIPULATIONs , it is out of sync with the automatically stored pre-aggregation identities, and many more identities are stored.

However, this is not the entire story. The numbers reported in Figure 4 were created using MANIPULATIONs that were instantaneous. This is rather deceptive, and fairly unlikely. In Figure 5, we show how each strategy performs with different length MANIPULATIONs. With fast MANIPULATIONs , on the order of 1 minute or less, Intermediate and Input Strategies do much better than the Strong Strategies. However, as the MANIPULATIONs get longer and near 10 minutes, the difference is less striking. When the MANIPULATIONs take an hour to run, the Strong Strategies are the clear winners with respect to time. This trend is because of the need to occasionally re-run MANIPULATIONs to get intermediate results for Intermediate and Input Strategies. Whenever a data item is reused by a different MANIPULATION in the Strong Strategies, it can be easily retrieved. However, in the Intermediate and Input strategies, the data item cannot be reused without being recomputed.

5 Discussion

Until now, we have merely explored the types of identification strategies available to users of implicit workflow systems. However, there are several open issues about *using* these identification strategies. While there are a myriad of possible topics, we discuss two distinct problems and lay out possible solutions below.

5.1 Identification Within an Implicit Workflow System

Consider that the problem of identification arises outside a workflow management system. As such, the implicit workflow is typically a cobbled together set of scripts, or

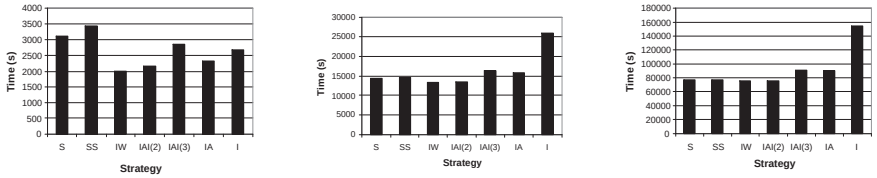


Fig. 5. The time for each identity strategy, particularly non-strong strategies, is related to how long it takes to re-run a manipulation. Manipulation execution times are set to a constant of (a) 1 minute, (b) 10 minutes, and (c) 1 hour.

worse, a user hitting "run", analyzing results, copying data, hitting "run" on another program, etc. How do you enforce proper data identification in these scenarios?

This is a difficult problem that requires either the automation of identification or a change in user behaviour. While a user may be willing to change enough to recognize that keeping proper identification is important, it is naïve to expect users to become organized, identifying, data custodians overnight. As such, some mechanism must be put into place that automatically provides data identification for disorganized users.

The logical answer to this problem is to teach the user how to use a workflow management system. Again, breaking all the bad-computation habits may be asking too much from the poor user. Is there a middle ground? A possible sketch would be a modified 'terminal' (e.g. xterm, terminal, dos-prompt), that looks the same as the user's default execution environment, but quietly keeps track in the background of the scripts run. Any identification strategy discussed above could be implemented behind such a system, transparent to the user.

5.2 Identification Across Disparate Workflow Systems

A separate problem occurs when dealing with data across multiple systems that were not designed to work together. For instance, a user wishes to integrate data produced through multiple distinct workflow systems. What identification strategy should be employed? While we see multiple possible methods for dealing with this, we shall discuss just one.

While using multiple workflow systems, if integration and homogeneous identification is the goal, then a possible strategy is utilizing one system as the "master" identifier. Anything from another system that is semantically or syntactically the same could be mapped to the same identifier in the "master" system. Anything from another system that cannot be mapped can be given a 'new' identification by the "master" system. Obviously there are pros and cons to this approach. We merely wish to point out an open problem, and provide the base for a discussion on it.

6 Related Work

Provenance needs, applications and types are incredibly diverse. There has been work studying lineage [7, 9, 8, 21], annotations [3, 27], capturing provenance [5, 20] and a range of forms, events, etc. Moreover, there has been work in creating provenance stores during workflow execution [1, 2, 4, 10, 17, 19, 24–26]. The Provenance Challenge [19] explored the requirements these workflow systems needed to adhere to in order to produce useful provenance stores.

Identity and Provenance needs in a workflow system were explored in [28]. This work built IDSet on top of the Taverna [26] workflow system. The workflow systems [1, 2, 4, 10, 17, 19, 24–26] are reliant upon the Strong identification strategy. This allows them to reuse intermediates data items and easily go back to intermediate states. Additionally, it enables them to distribute processes across the grid [11, 12, 18, 26] if applicable. Moreover, [2] explore identification strategies for entries in a cache that allow: data sharing, correct computation of the full object, and re-execution of only changed steps.

Finally, [5, 6] discuss enabling users to collect provenance records of their actions. By requiring users to utilize a particular tool, information is captured about user actions on provenance-unaware systems. This is similar to the strategy outlined in Section 5.1.

7 Conclusions

In this work, we focus on the provenance generated by "implicit" workflows; workflows created by a user with a specific goal, but outside a workflow framework. In particular, we study the needs for identification of data items in these systems. Unlike workflow systems [1, 2, 4, 10, 17, 19, 24–26] that are required to identify and keep all intermediate data items, many systems, [15, 16, 23] for example, do not need intermediate data. It is still imperative that they maintain provenance, and the provenance can be used to trace back to a data item's origin, but there is no need to keep intermediate data items.

We explore a set of identification strategies: Strong, Strong with IDSet, Intermittent and Input Only. Each of these identification strategies have strengths and weaknesses in terms of intermediate support, provenance capture time and storage space. We show that Strong identification is preferred not only for workflow style systems, but also for systems with long-running processes, where the "do over" time is large. On the other hand, we outline cases in which Intermittent and Input Only identification strategies would be preferable.

8 Acknowledgments

Thanks to Luc Moreau for his correspondence, which inspired this work. This work was supported in part by NSF grant number IIS 0741620 and by NIH grant number U54 DA021519.

References

1. Roger S. Barga and Luciano A. Digiampietri. Automatic capture and efficient storage of escience experiment provenance. In *Concurrency and Computation: Practice and Experience*, 2007.
2. L. Bavoil, S. Callahan, P. Crossno, J. Freire, C. Scheidegger, C. Silva, and H. Vo. VisTrails: Enabling interactive multiple-view visualizations. In *IEEE Visualization*, pages 18–26, 2005.
3. Deepavali Bhagwat et al. An annotation management system for relational databases. In *VLDB*, pages 900–911, 2004.
4. Shawn Bowers, Timothy McPhillips, Martin Wu, and Bertram Ludscher. Project histories: Managing data provenance across collection-oriented scientific workflow runs. In *DILS*, pages 27–29, 2007.
5. Peter Buneman, Adriane Chapman, and James Cheney. Provenance management in curated databases. In *ACM SIGMOD*, pages 539–550, June 2006.
6. Peter Buneman, Adriane Chapman, James Cheney, and Stijn Vansummeren. *A Provenance Model for Manually Curated Data*, pages 162–170. Lecture Notes in Computer Science. Springer Berlin / Heidelberg, Provenance and Annotation of Data edition, 2006.
7. Peter Buneman, Sanjeev Khanna, and Wang-Chiew Tan. Why and Where: A characterization of data provenance. In *ICDT*, pages 316–330, 2001.
8. Y. Cui, J. Widom, and J.L. Wiener. Tracing the lineage of view data in a data warehousing environment. In *ACM Transaction on Database Systems (TODS)*, 2000.
9. Yingwei Cui and Jennifer Widom. Lineage tracing for general data warehouse transformations. In *VLDB*, pages 41–58, 2001.
10. Luciano Digiampietri, Claudia Medeiros, and Joo Setubal. A framework based on web service orchestration for bioinformatics workflow management. *Genet. Mol. Res.*, 4(3):535–542, 2005.
11. Ian Foster, Jens Vockler, M Wilde, and Yong Zhao. The virtual data grid: a new model and architecture for data-intensive collaboration. In *CIDR*, 2003.
12. Paul Groth, Simon Miles, and Luc Moreau. PReServ: Provenance recording for services. In *Proceedings of the UK OST e-Science second All Hands Meeting 2005 (AHM'05)*, 2005.
13. Paul Groth, Simon Miles, and Luc Moreau. A Model of Process Documentation to Determine Provenance in Mash-ups. *Transactions on Internet Technology (TOIT)*, 2008.
14. James Howison, Andrea Wiggins, and Kevin Crowston. eResearch workflows for studying free and open source software development. In *IFIP 2.13*, 2008.
15. Magesh Jayapandian, Adriane Chapman, V.Glenn Tarcea, Cong Yu, Aaron Elkiss, Angela Ianni, Bin Liu, Arnab Nandi, Carlos Santos, Philip Andrews, Brian Athey, David States, and H.V. Jagadish. Michigan Molecular Interactions (MiMI): Putting the jigsaw puzzle together. *Nucleic Acids Research*, pages D566–D571, Jan 2007.
16. You Jung Kim, Andrew Boyd, Brian D. Athey, and Jignesh M. Patel. miBLAST: Scalable evaluation of a batch of nucleotide sequence queries with blast. *Nucleic Acids Research*, 33(13):4335–4344, 2005.
17. T. McPhillips, S. Bowers, and B. Ludscher. Collection-oriented scientific workflows for integrating and analyzing biological data. In *DILS*, pages 248–263, 2006.
18. Simon Miles, Paul Groth, Miguel Branco, and Luc Moreau. The requirements of recording and using provenance in e-science experiments. *Journal of Grid Computing*, 5(1):1–25, 2007.
19. Luc Moreau, Bertram Ludäscher, et al. The First Provenance Challenge. *Concurrency and Computation: Practice and Experience*, 2007. http://twiki.ipaw.info/bin/view/Challenge/SecondProvenanceChallenge .

20. Kiran-Kumar Muniswamy-Reddy, David A. Holland, Uri Braun, and Margo I. Seltzer. Provenance-aware storage systems. In *USENIX Annual Technical Conference*, pages 43–56, 2006.
21. Michi Mutsuzaki, Martin Theobald, et al. Trio-One: Layering uncertainty and lineage on a conventional DBMS. In *CIDR*, pages 269–274, 2007.
22. KD Pruitt, T Tatusova, and DR Maglott. NCBI reference sequence (RefSeq): a curated non-redundant sequence database of genomes, transcripts and proteins. *Nucleic Acids Research*, pages D501–D504, 2005.
23. Philip Resnik, Aaron Elkiss, Ellen Lau, and Heather Taylor. The web in theoretical linguistics research: Two case studies using the linguist's search engine. In *31st Meeting of the Berkeley Linguistics Society*, pages 265–276, February 2005.
24. Carlos E. Scheidegger, Huy T. Vo, David Koop, Juliana Freire, and Claudio Silva. Querying and re-using workflows with vistrails. In *SIGMOD*, 2008.
25. Yogesh Simmhan, Beth Plale, and Dennis Gannon. A framework for collecting provenance in data-centric scientific workflows. In *ICWS*, 2006.
26. Taverna, 2008. http://taverna.sourceforge.net/.
27. Y. Richard Wang and Stuart E. Madnick. A polygen model for heterogeneous database systems: The source tagging perspective. In *VLDB*, pages 519–538, 1990.
28. Jun Zhao, Carole Goble, and Robert Stevens. *An Identity Crisis in the Life Sciences*, pages 254–269. Lecture Notes in Computer Science. Springer Berlin / Heidelberg, Provenance and Annotation of Data edition, 2006.