Panda: A System for Provenance and Data*

Robert Ikeda
Stanford University
rmikeda@cs.stanford.edu

Jennifer Widom
Stanford University
widom@cs.stanford.edu

Abstract

Panda (for Provenance and Data) is a new project whose goal is to develop a general-purpose system that unifies concepts from existing provenance systems and overcomes some limitations in them. Panda is designed for "data-oriented workflows," fully integrating data-based and process-based provenance. Panda's provenance model will support a full range from fine-grained to coarse-grained provenance. Panda will provide a set of built-in operators for exploiting provenance after it has been captured, and an ad-hoc query language over provenance together with data. The processing nodes in Panda's workflows can vary from well-understood relational transformations, to "semi-opaque" transformations with a few known properties, to fully-opaque "black boxes." A theme in Panda is to take advantage of transformation knowledge when present, but to degrade gracefully when less information is available. Panda yields interesting optimization problems, including data caching decisions and eager vs. lazy provenance capture. This paper is largely an overview of motivation and plans for the project, with some material on current progress and results.

1 Introduction

In its most general form, *provenance* (also sometimes called *lineage*) captures where data came from, how it was derived, manipulated, and combined, and how it has been updated over time. Provenance can serve a number of important functions:

- **Explanation.** Users may be particularly interested in or wary of specific portions of a derived data set. Provenance supports "drilling down" to examine the sources and evolution of data elements of interest, enabling a deeper understanding of the data.
- **Verification.** Derived data may appear suspect—due to possible bugs in data processing and manipulation, because the data may be stale, or even due to maliciousness. Provenance enables auditing how data was produced, either for verifying its correctness, or for identifying the erroneous or outdated source data or processing nodes that are responsible for erroneous or outdated output data.
- **Recomputation.** Having found outdated or incorrect source data, or processing nodes that are buggy or have been modified, we may want to propagate corrections or changes to all "downstream" data that are affected. Provenance helps us recompute only those data elements that are affected by corrections or modifications.

Bulletin of the IEEE Computer Society Technical Committee on Data Engineering

*Work supported by the National Science Foundation under grants IIS-0414762 and IIS-0904497 and by the Boeing Corporation.

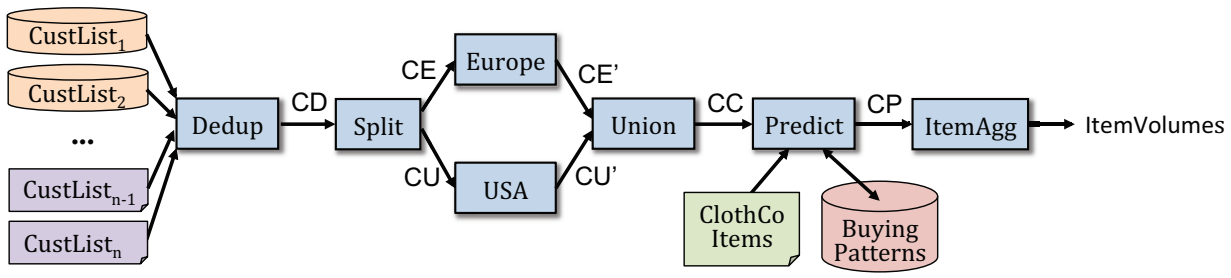

Figure 1: Analytics workflow example.

There has been a large body of very interesting work in lineage and provenance over the past two decades. Space limitations preclude detailed discussion of previous work here, but surveys are presented in, e.g., [2, 3, 5]. Despite all the past work, we believe there are still many limitations and open areas. Specifically:

1. Most work has been either: *data-based* (fine-grained), in which provenance of data elements is tracked based on well-defined, transparent properties of data models and query languages; or *process-based* (coarse-grained), in which provenance typically involves workflows and data at the schema level.
2. Often the primary focus is on *modeling* and *capturing* provenance: How is provenance information represented? How is it generated? There has been far less work on *querying* provenance: What can we do with provenance information once we've captured it?
3. Many projects have focused on specific functions or application domains, rather than developing a general provenance system that can be used for different purposes and across domains.

In the *Panda* (for "*Provenance and data*") project at Stanford, our goal is to fill these gaps. Specifically, in Panda we will:

1. Seamlessly merge data-based and process-based provenance, so that the two types of provenance can be combined (e.g., workflows that combine "opaque" processing nodes with well-understood relational queries and transformations). Thus, our model and system will offer users a full range from fine-grained to coarse-grained provenance.
2. Define a set of useful operators for taking advantage of provenance after it has been captured, as well as a general-purpose language for querying and analyzing provenance, and for combining provenance with relevant data.
3. Develop a general-purpose open-source system that is flexible and configurable enough to be used for a wide variety of applications. The system will support its own mechanisms for provenance capture, storage, operators, and queries, while also offering interfaces for coupling with outside data sources, processes, and systems.

2 Running Example

We use a detailed fictitious example to illustrate many of our goals for the Panda system. Consider *ClothCo*, a mail-order clothing company that is trying to decide which items to feature most prominently in its upcoming catalog. ClothCo runs an analytics workflow to predict which items will have the largest demand among its customers. The workflow is shown in Figure 1.

The initial input to the workflow is customer lists $\mathbf{CustList_1}$, $\mathbf{CustList_2}$, ..., $\mathbf{CustList_n}$, obtained from ClothCo's own databases and from partnerships with other companies. The lists contain names, addresses, and possibly additional attributes (e.g., gender, income), all of which are used for list deduplication and for buying-behavior predictions. The workflow involves the following steps:

- **Dedup**: Records from the customer lists are *deduplicated* so that customers represented in multiple lists are counted only once.
- **Split**: The deduplicated customer records (CD) are then partitioned into a European customer set (CE) and a USA customer set (CU). (If **Split** cannot determine that a customer is European, by default it assumes the customer is American.)
- **Europe / USA / Union**: Based on **Split**, records are routed either to **Europe** or **USA**, where the records' addresses are "canonicalized" into a standard form using existing software specialized for the region. The output lists CE' and CU' are unioned back to form canonicalized customer list CC.
- **Predict**: The next stage in the workflow predicts which ClothCo items customers are most likely to purchase. Specifically, for each customer in input list CC, and for each item ClothCo stocks, the **Predict** module consults a **Buying Patterns** database and attaches to the customer-item pair a likelihood that the customer will buy the item. **Predict**'s output CP is a set of customer-item-probability triples.
- **ItemAgg**: Finally, from the CP triples, **ItemAgg** aggregates the predicted demand for each of ClothCo's items. The output, ItemVolumes, contains a list of items and predicted sales volumes for each one.

Now suppose a ClothCo analyst runs the workflow and is surprised to find that the item in ItemVolumes predicted to have the highest demand is a cowboy hat, despite the fact that ClothCo's target customers rarely hail from the southern United States. Noticing this anomalous result, the analyst would like to find out why the predicted demand for cowboy hats is so high.

The Panda system will support the following interaction. Tracing the provenance of the cowboy-hat record in ItemVolumes back one step to CP yields a large data set, but with a simple provenance query the analyst discovers that the majority of the customers predicted to want the cowboy hat live in Paris, Texas. Something is clearly wrong: the population of Paris, Texas is only 25,000, and ClothCo doesn't cater to this demographic anyway. Further tracing the provenance of the relevant CP records, the analyst discovers that most of them were processed by **USA**, but they came originally from a French customer list. It turns out because the French list did not specify "France" explicitly in its addresses, **Split** mistakenly routed these customer records to **USA**, which added "Texas" during the canonicalization process. To fix the error, a simple new module is inserted that appends "France" to the addresses from the problematic list. Using provenance, when the modified workflow is rerun with the new module, only the item predictions potentially affected by the modifications are recomputed.

While this example captures a large number of the most important capabilities planned for the Panda system, there are a few missing. For example, we want to capture provenance from *human-generated* data edits and manipulations, along with computer-generated ones. Also, we expect that capturing and managing *evolving versions* of data, including long-term and frequent evolutions, will be a critical feature.

3 Processing Nodes and Provenance Capture

The workflow in our example is representative of the variety of processing node types that Panda will support. Specifically, processing nodes may vary from fully transparent (e.g., relational queries), to partially transparent (e.g., one-to-one transformations that apply an opaque function or filter on each element), to true "black boxes" (e.g., private code, calls to external services). Whenever possible, we want to capture provenance automatically. In addition, we plan to define an interface by which any type of processing node (including a human) can write out provenance information in a well-specified uniform fashion.

In our example, the **Union** and **ItemAgg** nodes perform standard relational operations. For relational nodes, there is a great deal of past work that can be applied for capturing and tracing provenance automatically and efficiently (see [3] for a survey). Consider **ItemAgg**, for example, which is a standard relational *group-by aggregation* operator. As defined by past work, the provenance of an element e output by **ItemAgg** consists of all elements in CP with the same *group-by* attribute as e.

Now consider **Dedup**, an opaque processing node. We cannot rely on the automatic methods for relational queries mentioned above to capture and trace provenance for **Dedup**. Either **Dedup** must be instrumented to write out provenance information as it executes (presumably in the form of mappings between deduplicated records and their original customer records), or it must provide some sort of procedure for provenance-tracing, or in the worst case we cannot determine fine-grained provenance at all.

One of the most important first steps in Panda is to formalize its provenance model. Possible models range from a simple bipartite graph structure connecting input and output data elements, to the much higher-level and more "semantic" *Open Provenance Model* [1]. We decided to begin with an implicit bipartite graph model induced by *provenance predicates* (Section 7). However, the model will need to be enhanced as the project develops, in keeping with our goals of combining data-based and process-based provenance, and capturing the range from fine-grained to coarse-grained provenance (recall Section 1). Based on the provenance model, Panda will specify a uniform interface by which all types of processing nodes can create and manipulate provenance, both manually and automatically.

Once provenance has been captured, what are we going to do with it? Our overall goal is to support the features mentioned in Section 1: explanation, verification, and recomputation. To do so, Panda will offer at least two methods of using provenance:

- A set of built-in *operations* that can be used on their own or as building blocks for higher-level functionality (Section 4)
- A full-featured *query language* that can be used to pose ad-hoc queries and analyses over provenance information, and over provenance information combined with relevant data (Section 5)

4 Provenance Operations

For basic built-in operations, Panda will support at least:

- **Backward tracing.** Given a derived data element e, where did e come from? That is, what data elements and/or processing contributed to e? In our running example, we used backward tracing to go from the output cowboy-hat record c to the record-set $\mathcal{R}$ in input list CP from which c was derived. Then we used further backward tracing to determine that most records producing $\mathcal{R}$ came through the **USA** processing node and originated from a specific French customer list.
- **Forward tracing.** Given an input or derived data element e, where did e subsequently go? That is, what processing nodes did e later pass through and what data elements were produced by it? In our running example, we can use forward tracing to determine all of the item predictions that were affected by French customers erroneously passing through the **USA** processing nodes.

From these two basic operations, we can layer on additional functionality, for example:

- **Forward propagation.** If an input or derived data element e changes, propagate the change to everything it affects. Clearly this function will rely on forward tracing. In our running example, once we correct the problem with our French customer list, we can use forward propagation to recalculate only those item predictions affected by the correction.
- **Refresh.** Given a derived data element e, check if e is still valid. If it is not, refresh it to its new valid value. Clearly this function will rely on both backward tracing and forward propagation. In our running example, suppose we correct the error with the French addresses, but ironically a celebrity then begins wearing cowboy hats. Suddenly cowboy hat sales soar, with corresponding modifications in our **Buying Patterns** database. We decide to once again investigate the cowboy hat prediction in ItemVolumes, this time asking for it to be refreshed to ensure we get the latest predicted demand.

5 Provenance Queries

The operations in the previous section are not specific to a particular application, but they still encode specific provenance-based functionality. We believe that in addition to a set of common operations, it will be very useful for a general-purpose provenance system to support a declarative ad-hoc query language, similar to what is provided by database management systems.

As we develop our query language, we have a few guiding principles. One is that we want our query language to operate over provenance and data seamlessly, so that they can be combined in useful ways. Second, we want our query language to be compact and intuitive for basic queries, with specific features for additional expressiveness, similar in spirit to SQL. Finally, the language must be amenable to finding efficient query execution plans, again in a database-system style.

Our running example relied on a query that aggregated the data elements in CP contributing to the cowboy-hat output element in ItemVolumes. Here are a few other queries that demonstrate the type of functionality we plan to support in a query language:

- Determine which customer list contributes the most to the top 100 predicted items.
- Considering only customers from a specific list, which items have significantly higher demand among those customers than among the general population?
- Which customers have more duplication in our original customer lists—those that are eventually processed by **USA**, or those that are processed by **Europe**?

6 System and Optimization Issues

Building a full-featured, general-purpose system for managing provenance and data together entails a significant effort just for the basic functions: provenance capture and storage, built-in operations like those in Section 4, and general-purpose query processing. (Progress on our initial prototype is described later in Section 7.2.) In addition, we see a number of specific opportunities and challenges, both theoretical and systems-related.

Query-Driven Capture

The most general approach is to capture provenance in support of any possible operation or query that may subsequently be performed using it. However, if there is a known restricted set of operations and queries to be performed, we may be able to streamline what is captured.

Eager vs. Lazy

Even when supporting arbitrary operations and queries, there may be a choice between *eager* and *lazy* provenance capture. It's a typical space-time tradeoff: eager capture occupies space but speeds up operations and queries. (Note that capturing all possible provenance at a fine-grained level could entail enormous amounts of space.) Also, similar to materialized views, eager capture may incur an update cost if we wish to keep provenance up-to-date. The choice between eager and lazy might be made on a per-processing-node basis. For example, eager capture makes sense for **Dedup**, since the overhead of recording the source customer records contributing to each deduplicated record may be fairly low, while recomputing that information may be very expensive (perhaps requiring deduplicating the entire data set again). On the other hand, for **ItemAgg**, if we have access to intermediate data set CP, then computing the provenance of an output element in **ItemAgg** is a simple selection query over CP.

Intermediate Results

Along similar lines, in a workflow such as our ClothCo example, we may or may not wish to store all of the intermediate data sets. Storing these data sets may be very helpful for provenance operations (such as finding the provenance of an **ItemAgg** output element by querying CP, as above) but the storage overhead may be high, and intermediate results may not always be necessary. For example, if we only wish to consider provenance from final output elements back to initial input data, and we take a fully eager approach, then no intermediate data sets are necessary. Overall, we envision a suite of interesting optimization problems involving decisions about intermediate data sets and eager vs. lazy computation.

Fine-Grained vs. Coarse-Grained

Another trade-off, perhaps intertwined with the previous two, is between fine-grained and coarse-grained provenance. Even if it is prohibitive to compute provenance eagerly at the data-element level, it may be helpful to record some provenance at the data-set level: some operations and queries may only need coarse-grained provenance, or perhaps coarse-grained provenance can be used to support later computation of fine-grained provenance. Suppose, for example, for each input list we record only the percentage of its customers that are eventually sent to processing node **Europe**, versus those sent to processing node **USA**. Queries may ask for this information directly, or in some cases (e.g., lists that are 100% **Europe** or 100% **USA**), this schema-level information may be used to optimize provenance queries involving individual records.

Approximation

Another avenue for dealing with prohibitively large fine-grained provenance may be to support *approximate* provenance capture, operations, and/or query results.

7 Initial Progress

We briefly describe some of our initial progress. We decided to drive our work by considering one of the specific operations discussed in Section 4, *refresh*. Targeting an "end-to-end" efficient solution to the refresh problem forced us to: become more concrete about the *data-oriented workflows* Panda is targeting; to develop formal underpinnings and a number of algorithms; and to code up a fledgling prototype system; all of which are now serving us well as a basis for the entire Panda project.

7.1 The Refresh Problem

Consider any workflow that is an acyclic graph of processing nodes, as in our running example. Suppose the input data sets have been modified since the workflow was run, but the workflow has not been rerun on the modified input. However, suppose we would like to *selectively refresh* one or more elements in the output data, i.e., compute the latest values of particular output elements based on the modified input data. With selective refresh, we can avoid the expense of recomputing an entire output data set frequently, when all we need is a few up-to-date output values after the input has undergone some changes.

Our initial approach to performing selective refresh assumes that we capture input-output data provenance at each processing node when the workflow is run initially, although lazy processing and optimizations like those discussed in Section 6 are certainly possible. In our initial approach, provenance takes the form of *provenance predicates*: Each output data element o has an associated predicate, which is applied to the input sets to compute o's provenance. For example, each element o in ItemVolumes has a provenance predicate that selects all customer-item-probability triples from CP with the same item as o. Provenance predicates are flexible and

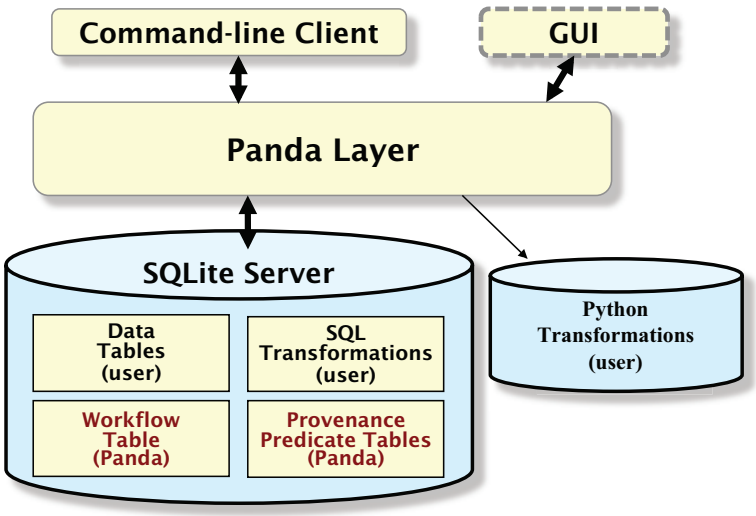

Figure 2: Architecture of the prototype Panda system.

general: Provenance in the form of pointers to specific data elements can be specified as predicates selecting on identifiers or keys, and provenance for basic relational operators [3] is easily expressed as predicates. Note that provenance predicates, when applied for all output elements, yield a bipartite graph structure as discussed in Section 3.

Frequently, provenance predicates take the same basic form for each data element in an output set, and we do plan to exploit that fact. For example, recently we added *attribute mappings* as a schema-level provenance specification mechanism. The provenance representation is very compact, while the semantics still logically yields a provenance predicate for each output element. For example, the provenance for ItemVolumes can be specified using an attribute mapping between CP.item and ItemVolumes.item. The element-level provenance predicates implied by this mapping are the same predicates discussed in the previous paragraph.

To perform refresh, we first apply provenance predicates recursively to determine which input elements are responsible for the output elements to be refreshed, i.e., we perform *backward tracing* as discussed in Section 4. Then we rerun the workflow only on that portion of the data needed for refresh. i.e., *forward propagation* from Section 4. Laying the formal foundations, developing the algorithms, and implementing refresh in a prototype system was a substantial challenge. One of our important results was to identify specific properties of processing nodes, provenance, and workflows for which refresh can be very efficient, while it is inherently less efficient when the properties do not hold. For details, please see our technical report [4].

7.2 Panda System Prototype

We began by building up the system infrastructure and features needed to experiment with all aspects of *refresh* as described in the previous section. In doing so, we produced an excellent starting point for the more ambitious and general-purpose Panda system.

The high-level architecture of the Panda system is shown in Figure 2. For now, all data must be representable as relational tables, and the main backend is a **SQLite** server. The server stores all data sets, relational (SQL) transformations, provenance, and workflow information. Panda also supports opaque processing nodes programmed in Python; they are stored separately in files.

Panda supports arbitrary acyclic graphs of processing nodes. Currently, users interact with Panda through a simple command-line interface; a GUI is underway. There are three general types of user commands: (1) Creating or modifying input tables; (2) Creating transformations that generate newly-defined tables from existing ones, to build up workflows; (3) Backward-tracing and refresh commands. When workflows are created and run,

Panda stores everything needed to support selective refresh: provenance and intermediate data sets for backward tracing, and transformations for forward propagation. Panda currently supports provenance predicates as described in Section 7.1; provenance based on schema-level attribute mappings (Section 7.1) is also underway.

Panda supports transformations specified as SQL queries, including queries involving multiple input tables. Provenance predicates are created automatically for SQL transformations, following known definitions and techniques [3]. For Python processing nodes, Panda currently supports one-one and one-many transformations only, so that provenance predicates can be generated automatically. Specifically, the "opaque" Python code associated with a processing node is run separately on each data element in the input set (multi-input Python transformations currently are not supported), producing zero or more elements for the output set. Input data elements are required to have keys, so the provenance predicate for an output element can simply select the key of the corresponding input element. Support for more general Python processing nodes is in progress.

Conclusions, Website, and Acknowledgments

The Panda project is in its early stages; much of the work is ahead of us. We have a fairly clear roadmap in terms of motivation, goals, and specific technical challenges, as laid out in Sections 1–6, and we have a solid start with the work reported in Section 7. For more information, and to follow progress in the Panda project, please visit our website: `http://i.stanford.edu/panda`.

We are grateful to Parag Agrawal, Charlie Fang, Diana MacLean, Abhijeet Mohapatra, Raghotham Murthy, Aditya Parameswaran, Hyunjung Park, Alkis Polyzotis, Semih Salihoglu, and Satoshi Torikai for many useful Panda meeting discussions that have been instrumental in forming the direction for our system. We thank Semih Salihoglu for his contributions to the refresh work and the Panda system, and we thank Charlie Fang and Satoshi Torikai for their contributions to the Panda system. We thank Philip Guo for his helpful comments on this paper.

References

[1] The Open Provenance Model (v1.01). July 2008. `http://eprints.ecs.soton.ac.uk/16148/`.

[2] BOSE, R., AND FREW, J. Lineage retrieval for scientific data processing: a survey. *ACM Comput. Surv. 37*, 1 (2005), 1–28.

[3] CHENEY, J., CHITICARIU, L., AND TAN, W.-C. Provenance in databases: Why, how, and where. *Foundations and Trends in Databases 1*, 4 (2009), 379–474.

[4] IKEDA, R., SALIHOGLU, S., AND WIDOM, J. Provenance-based refresh in data-oriented workflows. Technical report, Stanford University, 2010.

[5] SIMMHAN, Y. L., PLALE, B., AND GANNON, D. A survey of data provenance in e-science. *SIGMOD Rec. 34*, 3 (2005), 31–36.