PANDA
A System for Provenance and Data

Jennifer Widom — Stanford University
Example: Sales Prediction Workflow

CustList_1 → Dedup → Split → Union → Predict → ItemAgg → Item Sales

... → CustList_2 → Europe

CustList_{n-1} → USA

CustList_n → Split

Catalog Items → Buying Patterns
Example: Sales Prediction Workflow

CustList₁ → Dedup → Split → Europe → Union → Predict → ItemAgg → Backward Tracing → Item Sales

- CustList₂ → ... → CustListₙ

Europe → USA

Catalog Items → Buying Patterns

<table>
<thead>
<tr>
<th>Name</th>
<th>Address</th>
<th>Item</th>
<th>Prob</th>
<th>Demand</th>
</tr>
</thead>
<tbody>
<tr>
<td>Amelie</td>
<td>… Paris, Texas</td>
<td>Hat</td>
<td>.98</td>
<td>high</td>
</tr>
<tr>
<td>Pierre</td>
<td>… Paris, Texas</td>
<td>Hat</td>
<td>.98</td>
<td></td>
</tr>
<tr>
<td>Isabelle</td>
<td>… Paris, Texas</td>
<td>Hat</td>
<td>.98</td>
<td></td>
</tr>
</tbody>
</table>

Jennifer Widom
Example: Sales Prediction Workflow

<table>
<thead>
<tr>
<th>Name</th>
<th>Address</th>
<th>Name</th>
<th>Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>Amelie</td>
<td>65, quai d'Orsay, Paris</td>
<td>… Paris, Texas</td>
<td></td>
</tr>
<tr>
<td>Pierre</td>
<td>39, rue de Bretagne, Paris</td>
<td>… Paris, Texas</td>
<td></td>
</tr>
<tr>
<td>Isabelle</td>
<td>20, rue d’Orsel, Paris</td>
<td>… Paris, Texas</td>
<td></td>
</tr>
</tbody>
</table>
Example: Sales Prediction Workflow

<table>
<thead>
<tr>
<th>Name</th>
<th>Address</th>
<th>Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>Amelie</td>
<td>65, quai d’Orsay, Paris</td>
<td>Paris, France</td>
</tr>
<tr>
<td>Pierre</td>
<td>39, rue de Bretagne, Paris</td>
<td>Paris, France</td>
</tr>
<tr>
<td>Isabelle</td>
<td>20, rue d’Orsel, Paris</td>
<td>Paris, France</td>
</tr>
</tbody>
</table>

### Example: Sales Prediction Workflow

- **Forward Propagation**
  - CustList\(_1\)
  - CustList\(_2\)
  - \(\ldots\)
  - CustList\(_{n-1}\)
  - CustList\(_n\)

- **Dedup** → **Split** → **Europe** → **Union** → **Predict** → **ItemAgg** → **Item Sales**

- **Backward Tracing**

- **Catalog Items**
- **Buying Patterns**

### Example Table

<table>
<thead>
<tr>
<th>Item</th>
<th>Demand</th>
</tr>
</thead>
<tbody>
<tr>
<td>Beret</td>
<td>high</td>
</tr>
</tbody>
</table>
Provenance

Where data came from

How it was derived, manipulated, combined, processed, ...

How it has evolved over time
Uses for Provenance

**Explanation**
Sources and evolution of data; deeper understanding

**Debugging and Verification**
- Buggy or stale source data? Buggy processing?
- Error propagation paths
- Auditing

**Recomputation**
Propagate changes to affected “downstream” data
Some Application Domains

- Sales prediction workflows 😊
- Scientific-data workflows
  - Including human-curated data
  - Including evolving versions of data
- Any analytic pipeline
- “Extract-transform-load” (ETL) processes
- Information-extraction pipelines
Isn’t provenance the same thing as lineage?

Pretty much

Haven’t you worked on it before?

Yes

1. Data Warehousing project (long ago)
   - Lineage of relational views in warehouse: formal foundations, system/caching issues
   - Lineage in ETL pipelines: foundations & algorithms

2. “Trio” project (recently)
   - Data + Uncertainty + Lineage
   - Lineage primarily in support of uncertainty
Previous provenance work tends to be...

- Either data-based or process-based
  - Capture both: “data-oriented workflows”

- Either fine-grained or coarse-grained
  - Cover the spectrum in a unified fashion

- Focused on modeling and capturing provenance
  - Also support provenance operators and queries

- Geared to specific functions or domains
  - End with a general-purpose open-source system
Remainder of Talk

- Fundamentals
- Capturing provenance
- Exploiting provenance
- Concrete progress and results
- What’s next
Remainder of Talk

- Fundamentals
  - Data-oriented workflows
  - Provenance model
- Capturing provenance
- Exploiting provenance
- Concrete progress and results
- What’s next
Remainder of Talk

- Fundamentals
  - Data-oriented workflows
  - Provenance model

- Capturing provenance

- Exploiting provenance
  - Backward tracing & forward tracing
  - Forward propagation & refresh
  - Ad-hoc queries

- Concrete progress and results

- What’s next
Data-Oriented Workflows

- Graph of processing nodes; data sets on edges
- Assume (for now):
  - Statically-defined; batch execution; acyclic
- Don’t assume (for now):
  - Specific types of data sets or processing nodes
Processing Nodes

- **Goal**
  - Exploit knowledge and properties when present
  - Provide fallback when processing is opaque

- **Sample properties**
  - Known relational operator or query
  - Monotonic
  - One-many or many-one
  - *Map* function or *Reduce* function
Processing Nodes

- **General principle**
  - Stronger properties $\Rightarrow$ finer-grained input-output data relationships; more useful and efficient provenance

- **Sample properties**
  - Known relational operator or query
  - Monotonic
  - One-many or many-one
  - *Map* function or *Reduce* function
Processing Nodes: Example

- Known relational operators
- Many-one, nonmonotonic
- One-one, monotonic
- Opaque
Provenance Model

- **Ultimate goals:**
  - Support provenance at spectrum of granularities
  - Mesh data-oriented and process-oriented provenance
  - Composability/transitivity
  - Understandability

- **For now, simple underlying model:**
  - Mappings between input and output data elements
Provenance Capture

- Processing nodes provide provenance information along with output

Eager — generated at data-processing time versus
Lazy — “tracing procedure”
Relational operators — automatic, previous work, eager or lazy
Dedup — eager easy, lazy hard
One-ones — eager or lazy easy
Predict — it depends

Worst Case:
No access to fine-grained provenance
Provenance Operations — Basic

- **Backward tracing**
  Where did the Cowboy Hat record come from?

- **Forward tracing**
  Which sales predictions did Amelie contribute to?
Additional Functionality

- **Forward propagation**
  
  Update all affected predictions after customers move from Texas to France
Additional Functionality

- **Refresh**
  
  Get latest prediction for Cowboy Hat sales (only) based on modified buying patterns
  
  ≈ Backward tracing + Forward propagation
Provenance Queries

- How many people from each country contributed to the Cowboy Hat prediction?
- Which customer list contributed the most to the top 100 predicted items?
Provenance Queries

- For a specific customer list, which items have higher demand than for the entire customer set?
- Which customers have more duplication — those processed by USA or by Europe?
Provenance Queries

- For a specific customer list, which items have higher demand than for the entire customer set?
- Which customers have more duplication — those processed by USA or by Europe?

**Query language goals**
- Declarative ad-hoc queries à la database systems
- Seamlessly combine provenance and data
- Amenable to optimization
Concrete Progress and Results

1. Provenance predicates
   - Motivated by making *refresh* problem concrete
   - Drove initial Panda prototype

2. Attribute mappings

3. Generalized *map* and *reduce* workflows
   - Provenance capture, backward tracing, forward tracing, forward-propagation, refresh
   - Ad-hoc queries, optimizations
Concrete Progress and Results

1. Provenance predicates
   - Motivated by making *refresh* problem concrete
   - Drove initial Panda prototype

2. Attribute mappings

3. Generalized *map* and *reduce* workflows

Predicates $\supset$ Attribute Mappings $\supset$ GMRWs
Provenance Predicates

Provenance of output $o$ is $\sigma_p(I)$
Provenance Predicates

Provenance of output $o_i$ is $\sigma_{p_i}(I)$

- Worst case: $p_i = \text{TRUE}$
- Think: formalism to be instantiated
  - Predicates can have compact representations
  - Predicates can sometimes be generated automatically
- Natural recursive definition
- Extends to multiple inputs/outputs

Captures most existing provenance definitions
Selective Refresh Problem

Exploit provenance to efficiently compute the up-to-date value of selected output elements after the input (or processing nodes) may have changed.
Selective Refresh Problem

- Refreshing \( o_i \) through one processing node \( P \)
  1) Backward trace \( I^* = \sigma_{p_i}(I_{\text{new}}) \)
  2) Forward propagate \( o_{\text{new}} = P(I^*) \)
Selective Refresh Problem

- Refreshing $o_i$ through one processing node $P$

\[ I_{new} \rightarrow \text{ItemAgg} \rightarrow \begin{cases} \text{...} & \text{[ (Beret, medium), item='Beret' ]} \\
\text{...} & \text{Refresh} \end{cases} \]

$\sigma_{\text{item='Beret'}}$
Selective Refresh Problem

- Refreshing $o_i$ through the processing node $P$
  1) $I^* = \sigma_p (I_{new})$
  2) $o_{new} = P(I^*)$

Properties of processing nodes and their provenance:

Does this always "work"?
Does it make sense?
Selective Refresh Problem

- Refreshing $o_i$ through entire workflow
  1) Backward trace recursively
  2) Forward propagate through workflow

- Properties of workflow
  - Does this always “work”?
  - Does it make sense?
  - Is it efficient?
Selective Refresh Example

<table>
<thead>
<tr>
<th>Person</th>
<th>City</th>
<th>SalesE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Amelie</td>
<td>Paris</td>
<td>10</td>
</tr>
<tr>
<td>Pierre</td>
<td>Paris</td>
<td>10</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Person</th>
<th>City</th>
<th>SalesD</th>
</tr>
</thead>
<tbody>
<tr>
<td>Amelie</td>
<td>Paris</td>
<td>13</td>
</tr>
<tr>
<td>Pierre</td>
<td>Paris</td>
<td>13</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Person</th>
<th>City</th>
<th>SalesD</th>
</tr>
</thead>
<tbody>
<tr>
<td>Amelie</td>
<td>Paris</td>
<td>26</td>
</tr>
<tr>
<td>Pierre</td>
<td>Paris</td>
<td>13</td>
</tr>
<tr>
<td>Marie</td>
<td>Paris</td>
<td>30</td>
</tr>
</tbody>
</table>

$\text{Euros2$}\rightarrow\text{CitySum}$

<table>
<thead>
<tr>
<th>City</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>Paris</td>
<td>26</td>
</tr>
</tbody>
</table>

$\text{Person="Amelie"}$
$\text{Person="Pierre"}$

$\text{City="Paris"}$

<table>
<thead>
<tr>
<th>City</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>Paris</td>
<td>39</td>
</tr>
</tbody>
</table>

$\text{Person="Amelie"}$
$\text{Person="Pierre"}$

$\text{City="Paris"}$

<table>
<thead>
<tr>
<th>City</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>Paris</td>
<td>78</td>
</tr>
</tbody>
</table>

$\text{Person="Amelie"}$
$\text{Person="Pierre"}$
$\text{Person="Marie"}$

$\text{City="Paris"}$
**Required Properties**

*Requirement 3.1 (Provenance Correctness):* Consider any transformation instance $T(I) = O$ for $I \in \mathbb{I}_T$, and any $\langle o, p \rangle \in O$. Then $T(\sigma_p(I)) = \{\langle o, p \rangle\}$. □

*Requirement 3.2 (Provenance as Key):* Consider any transformation instance $T(I) = O$ for $I \in \mathbb{I}_T$, and any $\langle o, p \rangle \in O$. Then for any $I' \in \mathbb{I}_T$, if $T(\sigma_p(I')) \neq \emptyset$, then $T(\sigma_p(I')) = \{\langle o', p \rangle\}$ for some $o'$, and $\langle o', p \rangle \in T(I')$. □

*Requirement 4.1 (Workflow Safety):* Consider any workflow instance $(T_1 \circ T_2 \circ \ldots \circ T_n)(I_1) = I_{n+1}$. Every $T_i$ must be *safe with respect to* $T_{i-1}$, $i = 2..n$, defined as follows. Consider any $I'_{i-1} \in \mathbb{I}_{T_{i-1}}$. Let $I'_i = T_{i-1}(I'_{i-1})$. For any $\langle o, p \rangle \in T_i(I_i)$, we must have $\bigcup_{\langle o', p' \rangle \in \sigma_p(I_i)} T_{i-1}(\sigma_p(I'_{i-1})) = \sigma_p(I'_i)$. □
Panda System (version 0.1)

Command-line Client

Graphical Interface

Create Table

Panda Layer

SQLite

File System

Data Tables (user)

Workflow Table (Panda)

SQL Transformations (user)

Provenance Predicate Tables (Panda)

Forward Filter Tables (Panda)

Python Transformations (user)
Panda System (version 0.1)

Command-line Client

Graphical Interface

Create SQL Transformation

Panda Layer

SQLite

Data Tables (user)

Workflow Table (Panda)

SQL Transformations (user)

Provenance Predicate Tables (Panda)

Forward Filter Tables (Panda)

File System

Python Transformations (user)

Command Line Client

Graphical Interface

Create SQL Transformation

Panda Layer

SQLite

Data Tables (user)

Workflow Table (Panda)

SQL Transformations (user)

Provenance Predicate Tables (Panda)

Forward Filter Tables (Panda)

File System

Python Transformations (user)
Panda System (version 0.1)
Panda System (version 0.1)

Command-line Client

Graphical Interface

Panda Layer

SQLite

File System

Data Tables (user)

Workflow Table (Panda)

SQL Transformations (user)

Provenance Predicate Tables (Panda)

Forward Filter Tables (Panda)

Python Transformations (user)

Forward Trace
Panda System (version 0.1)

Command-line Client

Graphical Interface

Panda Layer

SQLite

Data Tables (user)

Workflow Table (Panda)

SQL Transformations (user)

Provenance Predicate Tables (Panda)

Forward Filter Tables (Panda)

File System

Python Transformations (user)

Refresh

Jennifer Widom
Attribute Mappings

Attribute mapping: \( I.A \leftrightarrow O.B \)

Provenance of output \( o \in O \) is: \( \sigma_{I.A=0.B}(I) \)

More generally: \( \forall x: \sigma_{B=x}(O) = P(\sigma_{A=x}(I)) \)

I (cust, item, prob) \( \rightarrow \) ItemAgg \( \rightarrow \) O (item, sales) \( \leftrightarrow \) I.Item \( \leftrightarrow \) O.item
Attribute Mappings

Attribute mapping: $I.A \leftrightarrow O.B$

Provenance of output $o \in O$ is: $\sigma_{I.A=o.B}(I)$

More generally: $\forall x: \sigma_{B=x}(O) = P(\sigma_{A=x}(I))$

- Can generate automatically in many cases (e.g., SQL)
- Worst case: $\{ \} \leftrightarrow \{ \}$
- Generalize to Datalog-like rules

$I(\_, \text{item}, \_): - O(\text{item}, \_)$

$I(\_, \text{item}, \text{prob}): - O_1(\text{item}, \_)$ \text{prob} > .95

$I(\_, \text{item}, \text{prob}): - O_2(\text{item}, \_)$ \text{prob} \leq .95

- Allow functions

$I.name \leftrightarrow \text{ToCaps}(O.name)$
Attribute Mappings

Attribute mapping: \( I.A \leftrightarrow O.B \)

Provenance of output \( o \in O \) is: \( \sigma_{I.A=o.B}(I) \)

More generally: \( \forall x: \sigma_{B=x}(O) = T(\sigma_{A=x}(I)) \)

- Rules for AMs: combining, splitting, transitivity
  - soundness and completeness
  - “Strongest possible mapping”
Provenance Operations

- Backward and forward tracing
- Forward propagation and refresh
- Key challenge: “broken chains”
- Proofs of correctness and minimality
Generalized Map and Reduce Workflows

What if every transformation was a Map or Reduce function?

- Very specific properties
- Provenance easier to define, capture, and exploit
- Automatic wrapping, doesn’t interfere with parallelism
Map and Reduce Provenance

- **Map functions**
  - $M(I) = \bigcup_{i \in I} (M\{i\})$
  - Provenance of $o \in O$ is $i \in I$ such that $o \in M\{i\}$

- **Reduce functions**
  - $R(I) = \bigcup_{1 \leq k \leq n} (R(I_k))$  $I_1, \ldots, I_n$ partition $I$ on reduce-key
  - Provenance of $o \in O$ is $I_k \subseteq I$ such that $o \in R(I_k)$
Recursive MR Provenance

- Intuitive recursive definition

  Workflow $W$ with inputs $I_1, \ldots, I_n$; output element $o$

  $$P_W(o) = (I^*_1, \ldots, I^*_n) \quad I^*_1 \subseteq I_1, \ldots, I^*_n \subseteq I_n$$

- Desirable property

  $$o \in W(I^*_1, \ldots, I^*_n)$$

  Usually holds, but not always
Counterexample

Twitter Posts

TweetScan

Inferred Movie Ratings

Summarize

Rating Medians

Count

#Movies Per Rating

“Avatar was great”

“I hated Twilight”

“Twilight was pretty bad”

“I enjoyed Avatar”

“I loved Twilight”

“Avatar was okay”

<table>
<thead>
<tr>
<th>Movie</th>
<th>Rating</th>
</tr>
</thead>
<tbody>
<tr>
<td>Avatar</td>
<td>8</td>
</tr>
<tr>
<td>Twilight</td>
<td>0</td>
</tr>
<tr>
<td>Twilight</td>
<td>2</td>
</tr>
<tr>
<td>Avatar</td>
<td>7</td>
</tr>
<tr>
<td>Twilight</td>
<td>9</td>
</tr>
<tr>
<td>Avatar</td>
<td>4</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Movie</th>
<th>Median</th>
</tr>
</thead>
<tbody>
<tr>
<td>Avatar</td>
<td>7</td>
</tr>
<tr>
<td>Twilight</td>
<td>2</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Median</th>
<th>#Movies</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>7</td>
<td>1</td>
</tr>
</tbody>
</table>
TweetScan

Inferred Movie Ratings

Summarize

Rating Medians

Count

#Movies Per Rating

Twitter Posts

“Avatar was great”

“I hated Twilight”

“Twilight was pretty bad”

“I enjoyed Avatar”

“I loved Twilight”

“Avatar was okay”

<table>
<thead>
<tr>
<th>Movie</th>
<th>Rating</th>
</tr>
</thead>
<tbody>
<tr>
<td>Avatar</td>
<td>8</td>
</tr>
<tr>
<td>Twilight</td>
<td>0</td>
</tr>
<tr>
<td>Twilight</td>
<td>2</td>
</tr>
<tr>
<td>Avatar</td>
<td>7</td>
</tr>
<tr>
<td>Twilight</td>
<td>9</td>
</tr>
<tr>
<td>Avatar</td>
<td>4</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Movie</th>
<th>Median</th>
</tr>
</thead>
<tbody>
<tr>
<td>Avatar</td>
<td>7</td>
</tr>
<tr>
<td>Twilight</td>
<td>2</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Median</th>
<th>#Movies</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>1</td>
</tr>
</tbody>
</table>
Counterexample

TweetScan

Inferred Movie Ratings

Summarize

Rating Medians

Count

#Movies Per Rating

Twitter Posts

“Avatar was great”

“I hated Twilight”

“Twilight was pretty bad”

“I enjoyed Avatar And Twilight too”

“Avatar was okay”

<table>
<thead>
<tr>
<th>Movie</th>
<th>Rating</th>
</tr>
</thead>
<tbody>
<tr>
<td>Avatar</td>
<td>8</td>
</tr>
<tr>
<td>Twilight</td>
<td>0</td>
</tr>
<tr>
<td>Twilight</td>
<td>2</td>
</tr>
<tr>
<td>Avatar</td>
<td>7</td>
</tr>
<tr>
<td>Twilight</td>
<td>7</td>
</tr>
<tr>
<td>Avatar</td>
<td>4</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Movie</th>
<th>Median</th>
</tr>
</thead>
<tbody>
<tr>
<td>Avatar</td>
<td>7</td>
</tr>
<tr>
<td>Twilight</td>
<td>2</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Median</th>
<th>#Movies</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>7</td>
<td>1</td>
</tr>
</tbody>
</table>
THEOREM 3.1. Consider a GMRW $W$ composed of transformations $T_1, \ldots, T_n$, with initial inputs $I_1, \ldots, I_m$. Let $o$ be any output element, and consider $P_W(o) = (I_1^*, \ldots, I_m^*)$.

1. If all map and reduce functions in $W$ are one-one or many-one, respectively, then $o = W(I_1^*, \ldots, I_m^*)$. (Note this result is stronger than the general $o \in W(I_1^*, \ldots, I_m^*)$.)

2. If there is at most one nonmonotonic reduce function in $W$, then $o \in W(I_1^*, \ldots, I_m^*)$. □
RAMP System

- Built on top of Hadoop, experiments on EC2
- Proof of concept — very preliminary!
- Wrap Map and Reduce functions (automatically) to capture provenance
  - Add \((file, offset)\) as IDs to output sets
  - Backward-tracing bias
  - Alternative schemes, indexing
  - Overhead on example: \(\sim 111\%\) time, \(\sim 45\%\) space
- Straightforward backward-tracing
  - Seconds response time on 1.2GB workflow
What’s Next

- Unify what we have so far
  Predicates $\supset$ Attribute Mappings $\supset$ GMRWS

- Enhance system(s)

- Extend provenance model
  - Fine-grained to coarse-grained
  - Data-based and process-based
  - Time/versioning
  - Extensions to capture, tracing, propagation
What’s Next

- Ad-hoc queries
  - Language
  - Execution
  - Optimization
  - Query-driven provenance capture
What’s Next

- Computation and storage optimizations
  - Eager vs. lazy provenance capture
    - Space-time & query-update tradeoffs
    - Processing-node dependent
      - Example: Retain intermediate data sets?
    - Extreme case
      - Workflow run once, never updated
      - Provenance traced frequently
      - Compute transitive provenance eagerly, discard intermediate data
What’s Next

- Provenance optimizations
  - Fine-grained vs. coarse-grained
  - Approximate provenance
PANDA
A System for Provenance and Data

“stanford panda”