Provenance in the Wild

4th USENIX Workshop on the Theory and Practice of Provenance

TaPP ’12

JUNE 14—15, 2012
BOSTON, MA

sponsored by USENIX in cooperation with ACM SIGOPS, ACM SIGMOD, and ACM SIGPLAN

Peter Macko, Margo Seltzer

June 14, 2012
What’s the Problem?

• What does it mean to collect provenance when you don’t control:
  – The data (types, format, organization, structure)
  – The operators
  – The environment in which its processed

• Can you impose/extract any semantic meaning to provenance when it’s collected by a herd of cats?

What do the Cats do?

• They use data in arbitrary formats
  – Flat files
  – Unstructured, semi-structured, badly-structured
  – Proprietary formats
  – The cram twelve different kinds of data into a single container.

• Transformations are arbitrary code
  – Pick your favorite turing-complete language.
  – Apply said language to data.
  – Transformations can depend on the environment.
  – Repeat

• They move data around
  – Download objects from the web
  – Copy, rename objects
  – Replace objects

• They install new software
  – New programs
  – New libraries
  – New compilers
A Proposed Architecture

Applications
In multiple languages

Language adapters

Provenance Library
C++

Database adapters

Provenance Store
With multiple implementations

June 2011
Why do we think this is a good idea?

• Heterogeneous environments are the norm.
• Provenance must span those environments.
• Users and/or applications can:
  – create connections that are implicit or unobservable by software systems.
  – Integrate both static and dynamic dependencies.

Bring provenance to the users rather than the users to the provenance.
Basic Use Model

• Connect to the library: cpl_attach
• Disclose provenance
  – Create/lookup objects: cpl_create_object, cpl_lookup_object
  – Disclose data flow: cpl_data_flow
  – Disclose control flow: cpl_control_flow
  – Add properties to objects: cpl_add_property
• Disconnect from the library: cpl_detach
Naming

• Goal is to allow interoperability with minimal coordination.

• Objects are identified by three parameters:
  – Namespace: the application or system component that “owns” the object. Examples: OS, a specific database, workflow engine or application, or a project.
  – Name: local name (unique within a namespace)
  – Type: file, process, or namespace-specific type
  – Version: cycle avoidance algorithm create versions
Additional Automatic Capture

- Capture object creation MAC address so that we can transmit provenance across a network (and still identify it).
- Capture provenance of provenance
  - Ties provenance to a specific instance of an application (e.g., a process).
  - Results in capture of command line arguments (e.g., size of the Java heap).

June 2012
Use Case: GraphDB Bench

- A benchmark suite (and lots of experiments) to evaluate absolute and relative performance of graph databases.
- Instrument flow from the graph database to the benchmark operators to results.
- Modifications: 270 lines of code (out of 7500 total)
  - Most is cut and paste
- Result: every csv result file has provenance indicating which operations were run, what the source database was, etc.
- Helped us debug benchmark suite, identify missing benchmark results, etc.
- Integration with scripts led us to develop command-line tool to track directory creation, file copies, etc.
Discussion

• Won’t this free for all lead to semantically meaningless provenance?
  – Some provenance is better than no provenance.
  – Users/application developers who care are likely to provide more semantically meaningful provenance than is available by less flexible systems.

• What do you do about missing provenance?
  – Some provenance is better than no provenance.
  – “Downstream” applications can connect upstream to bypass provenance oblivious applications.

• Bottom line: We make rope – make it possible to have provenance without requiring that analysts or programmers use specific languages or tools.