Using Provenance for Repeatability

Quan Pham\textsuperscript{1}, Tanu Malik\textsuperscript{2}, Ian Foster\textsuperscript{1,2}

\textit{Department of Computer Science}\textsuperscript{1,\$} and \textit{Computation Institute}\textsuperscript{2,¶}

\textit{University of Chicago}\textsuperscript{\$¶} and \textit{Argonne National Laboratory}\textsuperscript{¶}

TaPP 2013
Publication Process

- Traditional academic publication process
  - Authors: Submit paper
  - Peer Reviewers: Review ideas & experiments
  - Readers: Learn novel methods.

- Emerging academic publication process
  - Authors: Submit paper
  - Peer Reviewers / Testers: Review ideas & experiments
    - Validate software
  - Readers: Are we reading something that is repeatable and reproducible?
Repeatability Testing

- Scientific progress relies on novel claims and verifiable results
- Scientific paper reviewers
  - Validate announced results
  - Validate for different data and parameters
  - Validate under different conditions and environments
- **Challenge:** Work under time & budget constraints

Image: from http://catsandtheirmews.blogspot.com/2012/05/update-on-computer-crash.html
Repeatability Testing
Challenges & Constraints

- Repeatability requirements
  - Hardware: Single machine/Clusters
  - Software
    - Operating System: Which operating system was used? (Ubuntu/RedHat/Debian)
    - Environment: How to capture all environment variables?
    - Tools & libraries installation: How to precisely know all the dependencies?

- Knowledge constraints
  - Experiment setup: how to setup the experiment?
  - Experiment usage: how the experiment is run?

- Resource constraints
  - Requires massive processing power.
  - Operates on large amounts of data.
  - Performs significant network communication.
  - Is long-running.
## An Approach to Repeatability Testing

<table>
<thead>
<tr>
<th>Challenges &amp; Constraints</th>
<th>Possible Solutions</th>
</tr>
</thead>
</table>
| • Repeatability requirements  
  • Hardware requirement  
  • Software requirement | • Provide a virtual machine  
  • Provide a *portable* software |
| • Knowledge constraints  
  • Experiment setup  
  • Experiment usage | Provide a reference execution |
| • Resource constraints | Provide selective replay |
PTU – Provenance-To-Use

- **PTU**
  - Minimizes computation time during repeatability testing
  - Guarantees that events are processed in the same order using the same data

- **Authors build a package that includes:**
  - Software program
  - Input data
  - Provenance trace

- **Testers may select a subset of the package’s processes for a partial deterministic replay**
PTU Functionalities

• *ptu-audit* tool
  • Build a package of authors’ source code, data, and environment variables
  • Record process- and file-level details about a reference execution
    ```
    % ptu-audit java TextAnalyzer news.txt
    ```

• PTU package
  • Display the provenance graph and accompanying run-time details

• *ptu-exec* tool
  • Re-execute specified part of the provenance graph
    ```
    % ptu-exec java TextAnalyzer news.txt
    ```
ptu-audit

- Uses `ptrace` to monitor system calls
  - `execve`, `sys_fork`
  - `read`, `write`, `sys_io`
  - `bind`, `connect`, `socket`
- Collects provenance
- Collects runtime information
- Makes package
ptu-audit

- Use ptrace to monitor system calls
  - execve, sys_fork
  - read, write, sys_io
  - bind, connect, socket
- Collect provenance
- Collect runtime info
- Make package
PTU Package

Figure 2. The PTU package. The tester chooses to run the sub-graph rooted at `/bin/calculate`. 

Provenance DB

- `/data/conf.dat`
  - `read()`

- `/data/in.dat`
  - `read()`
  - `fork()`

- `/bin/filter`
  - `write()`
  - `fork()`

- `/bin/convert`

- `/bin/calculate`
  - `fork()`

- `/bin/workflow`
  - `fork()`

- `$PKG_ROOT/bin/
  - workflow`
  - `reclass`
  - `calculate`
  - `filter`
  - `convert`
  - `python`

- `/data/conf.dat`
- `/data/in.dat`
- `/data/out.dat`
ptu-exec

- **Figure 3.** ptu-exec re-runs part of the application from /bin/calculate. It uses CDE to re-route file dependencies.
Current PTU Components

- Uses CDE (Code-Data-Environment) tool to create a package
  - CDE is a tool to package code, data, and environment required to deploy and run your Linux programs on other machines without any installation or configuration

- Uses **ptrace** to create a provenance graph representing a reference run-time execution

- Uses SQLite to store the provenance graph

- Uses **graphviz** for graph presentation

- Enhances CDE to run the package

Get Data
- Wget
- Bash script

Reclassify
- R
- Raster
- Rgdal
- Reclassify

Calculate
- R
- Geo algorithm
3-step workflow with 7 classifications

3-step workflow replayed in PTU with 1 classification
TextAnalyzer


- runs a named-entity recognition analysis program using several data dictionaries

- splits the input file into multiple input files on which it runs a parallel analysis
Figure 5. Time reduction in testing TextAnalyzer using PTU.

- No Monitoring
- PTU audited for provenance & package
- Replay with lower convergence criteria

Runtime (seconds):
- Real time: 72.035
- User time: 28.902
- System time: 13.408
Conclusion

• PTU is a step toward testing software programs that are submitted to conference proceedings and journals to conduct repeatability tests

• Easy and attractive for authors

• Fine control, efficient way for testers
Future Works

• Other workflow type
  • Distributed workflows.

• Improve performance
  • Decide how to store provenance compactly in a package.

• Presentation
  • Improve graphic-user-interface and presentation
Acknowledgements

• Neil Best
• Jonathan Ozik
• Center for Robust Decision making on Climate and Energy Policy (NSF grant number 0951576)
• Contractors of the US Government under contract number DEAC02-06CH11357