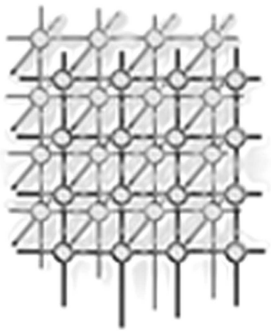
CONCURRENCY AND COMPUTATION: PRACTICE AND EXPERIENCE
Concurrency Computat.: Pract. Exper. 2007; **0**:1–10 *Prepared using* cpeauth.cls *[Version: 2002/09/19 v2.02]*

Automatic capture and reconstruction of computational provenance



James Frew*, Dominic Metzger, Peter Slaughter

Donald Bren School of Environmental Science and Management, University of California, Santa Barbara, CA 93106-5131, USA

SUMMARY

The Earth System Science Server (ES3) project is developing a local infrastructure for managing Earth science data products derived from satellite remote sensing. By "local," we mean the infrastructure that a scientist uses to manage the creation and dissemination of her own data products, particularly those that are constantly incorporating corrections or improvements based on the scientist's own research. Therefore, in addition to being robust and capacious enough to support public access, ES3 is intended to be flexible enough to manage the idiosyncratic computing ensembles that typify scientific research.

Instead of specifying provenance explicitly with a workflow model, ES3 extracts provenance information automatically from arbitrary applications by monitoring their interactions with their execution environment. These interactions (arguments, file I/O, system calls, etc.) are logged to the ES3 database, which assembles them into provenance graphs. These graphs resemble workflow specifications, but are really reports - they describe what actually happened, as opposed to what was requested. The ES3 database supports forward and backward navigation through provenance graphs (i.e. ancestor/descendant queries), as well as graph retrieval.

KEY WORDS: ES3, instrumentation, lineage, passive, provenance, transparency

1. INTRODUCTION

Scientists are increasingly being called upon to publish data as well as conclusions [1]. Computational science, in particular, often involves the creation of data products as a primary goal, rather than simply a means to an end. The "value added" in these products is primarily

*Correspondence to: frew@bren.ucsb.edu
Contract/grant sponsor: National Aeronautics and Space Administration; contract/grant number: NNG04GC52A and NNG04GE66G

Received 29 November 2006
Revised 08 March 2007

a function of their embodying the scientists most current understanding of the problem. The scientist is therefore motivated to make the transition from experimental to operational (i.e., disseminated) data product as seamless as possible.

Decades of experience with developers of data products based on satellite remote sensing of snow cover [2] and ocean color [3] have convinced us that most practicing scientists are extremely reluctant to assume the burdens of operational data publication. The primary reasons for this are:

- Scientists are usually funded to do research, not to maintain operational data dissemination systems. Computing infrastructures built to support research and development are frequently ill suited to publication and dissemination.
- Computational earth science is frequently conducted in advanced analysis environments (e.g., software packages) that scientists spend a great deal of time mastering and extending, and are therefore reluctant to abandon for a "production" computing environment.

The Earth System Science Server (ES3) directly addresses these challenges, by providing a local (to the scientist) computing infrastructure that seamlessly supports both exploratory computational science (specifically, scientific algorithm development) and operational product generation and dissemination (specifically, web access to low-cost, scalable computing and storage servers.) Common to both of these activities is the acquisition, maintenance, and publication of metadata about the evolving algorithms and resulting products. This paper describes the ES3 solution to capturing, reconstructing, and visualizing provenance metadata.

2. PROVENANCE IN ES3

2.1. Strategy

The primary challenge posed to provenance management by the ES3 environment is the requirement that the scientist be allowed to use whatever software tools she is most comfortable with. ES3 therefore adopts a novel approach to provenance capture and management: rather than requiring that scientists express their algorithms, procedures, or workflows using a specific language, system, or standard, ES3 instead transparently captures provenance-related information at execution time, from whatever analysis software the scientist happens to be using. The captured information is then assembled *post hoc* into a provenance graph (i.e., linked inputs, processes, and outputs.)

Broadly speaking, there are three strategies available for capturing provenance at execution time [4]:

1. **Passive monitoring**: This strategy entails tracing a processs interaction with its environment. We call this "passive" since it involves no modifications to either the process or its execution environment. All provenance relationships must be deduced *post hoc*.
2. **Overriding**: This strategy entails replacing portions of a processs execution environment (e.g., shared libraries) with instrumented versions that explicitly capture provenance

Figure 1. ES3 Components

information. As in passive monitoring, no modification of the science process itself is required, although detailed knowledge of (and access to) the science execution environment is required.

3. **Instrumentation**: This strategy entails inserting specific provenance capture instructions into the science process. Detailed knowledge of (and access to) the science process source code is required.

The strategies are ranked in order of both increasing intrusiveness into the scientific programming environment, and increasing selectivity as to the information acquired (e.g., everything else being equal, instrumentation is likely to acquire less extraneous information than passive monitoring.) The ES3 provenance system utilizes all of these strategies, individually or in combination, depending on the specific science codes being executed and the degree of access available.

2.2. Architecture and Implementation

Provenance in ES3 is managed by two components: the Probulator and the ES3 Core (Figure 1.)

The ES3 Probulator is designed to non-intrusively monitor the execution of complex scientific applications. All operations of the Probulator are completely transparent to ES3 users, and the default mode of operation requires no modification whatsoever of existing codes.

The Probulator comprises two applications, the Logger and the Transmitter. Both applications are user-mode non-privileged processes that are (usually) started automatically.

The Logger automatically instruments, monitors, and logs the execution of targeted programs and their interactions with their environment (files, parameters, system calls, etc.) "Plugins" adapt the Logger to different scientific processing environments. Currently two plugins are provided:

1. The default plugin uses system call tracing to intercept and log a subset of the probulated process's system calls; e.g.:

```
...
22636 1157750059.747511 execve("/AIR5.2.5/bin/align_warp", ...
22636 1157750059.810691 open("/etc/ld.so.cache", O_RDONLY) = 3
22636 1157750059.811162 open("/lib/libm.so.6", O_RDONLY) = 3
22636 1157750059.811638 open("/lib/libc.so.6", O_RDONLY) = 3
22636 1157750059.813693 open("anatomy1.hdr", O_RDONLY) = 3
22636 1157750059.814905 open("reference.hdr", O_RDONLY) = 3
...
```

Since system call traces can be voluminous, a default configuration file specifies system call patterns (e.g., shared library accesses) that can safely be ignored. This plugin currently works on Linux (and should work on any UNIX-like system that supports the `strace` facility.) When using this plugin, the Logger is invoked as a Linux command that spawns a traced shell to run the probulated process, and collects the shell's trace information.

2. A plugin for the IDL [5] analysis environment preprocesses IDL scripts to insert ES3 specific logging information, and to replace calls to certain IDL built-in functions with calls to instrumented ES3 equivalents. Although this plugin does modify the targeted application code, it does so transparently and reversibly – no user intervention is required beyond setting a flag in an environment variable to enable or disable probulation. When using this plugin, the Logger is invoked as a co-process by the user's IDL startup routine.

Upon termination of a Logger session (or on specific request), Logger log files:

```
...
<exec time="20060908T211419.747511Z" routine="/AIR5.2.5/bin/align_warp"
      pid="22636">
<io>
  <pipe read="true" id="std-in"/>
  <pipe write="true" id="std-out"/>
  <pipe write="true" id="std-err"/>
  <file read="true">/etc/ld.so.cache</file>
  <file read="true">/lib/libm.so.6</file>
  <file read="true">/lib/libc.so.6</file>
  <file read="true">/home/es3/provenance_challenge/anatomy1.hdr</file>
  <file read="true">/home/es3/provenance_challenge/reference.hdr</file>
</io>
...
```

are read by the Transmitter, which:

1. assigns a universally unique identifier (UUID) to every provenance-relevant object (file or process) referenced in the log file;
2. converts the plugin-specific log files into standard ES3 execution reports; and
3. sends these reports as XML messages:

```
...
<ES3Request type="registerFile">
  <file>
    <workflowUuid>5f04fceb-4691-4c8c-b5c3-96b9449eccf0</workflowUuid>
    <domain name="Local file system">
      <identifier>/home/es3/provenance_challenge/anatomy1.hdr</identifier>
    </domain>
    <uuid>c9ba5bd5-98a4-437c-af12-dd64f42fe57d</uuid>
  </file>
```

```
</ES3Request>
...
```

via a web service interface to the ES3 Core.

The ES3 Core decomposes the execution reports into object references and linkages between objects, using the Transmitter-supplied UUIDs as primary keys. This allows the Core to reconstruct the provenance graph at arbitrary starting points, forward and backward in time, by following the UUID references. The Core can also use file name, process name, and argument information captured by the Probulator to map between UUIDs and external names, allowing ES3 users to form queries in terms of objects they're familiar with.

2.3. ES3 Contributions to Provenance Challenge

- *Execution Environment*: ES3 can capture the provenance of any arbitrary sequence of Linux processes. For the challenge, provenance information was captured by passive monitoring: the `bash` shell executing the workflow had system call tracing enabled and redirected to the appropriate Logger plugin.
- *Representation Technology*: ES3 represents provenance as XML documents[†]. The ES3 Core stores provenance information in an XML database.
- *Query Language*: The ES3 Core implements a set of custom XML provenance requests that include XQuery constraint expressions.
- *Research Emphasis*: ES3 is focuses on recording, storing, and querying provenance information. ES3 is not an execution environment.
- *Challenge Implementation*: We executed the challenge workflow shell script directly, without any modifications.
- *Includes Workflow Representations*: ES3 is not a workflow environment; any workflow information is deduced *post hoc*.
- *Data Derivation vs. Causal Flow of Events*: ES3 extracts provenance information from event traces, which in most cases include all relevant data accesses, so the captured provenance can be viewed from either perspective.
- *Annotations*: The Probulator used for the challenge did not handle annotations. (We discuss this further in Section 4.)
- *Time*: The Probulator uses local (system clock) time information to order the execution trace information, and the time information is passed on to the ES3 Core and may be queried.
- *Naming*: The ES3 Core constructs lineage graphs by matching up UUIDs inserted into the provenance data by the Transmitter.
- *Tracked data, Granularity*: Since ES3 tracks provenance at the system call level, it is theoretically capable of recording the provenance of individual data bytes. Currently, the granularity is restricted to file open/close and process execute/exit events.

[†]RELAX NG schema: http://eil.bren.ucsb.edu/ES3/SecondProvenanceChallenge/getLineage-schema.rng

Figure 2. Provenance Challenge workflow reassembled by ES3

- *Abstraction Mechanisms*: ES3's notion of a "workflow" corresponds to a single execution trace (e.g. of a shell script.) Otherwise, ES3's abstractions correspond directly to files and processes.

2.4. Comparison with other Provenance Challenge approaches

Most approaches represented in the Provenance Challenge [6] were based on workflow systems [7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17]. These systems all employ a *prescriptive* approach to provenance: all computations are described by a workflow model that is semantically, and in many cases syntactically, close or identical to the provenance graph. Put another way, most of the work involved in provenance collection is done "up front," when the workflow is specified. Collection of instance-level provenance information can also be simplified, by building it into the workflow environment.

The common disadvantage of the workflow approaches is the requirement that computations be restricted to those supported by or expressible in a specific workflow system. In addition to limiting the scope of possible computations, this requires users of these systems to master and then exclusively use a specific workflow creation environment.

By contrast, both ES3 and PASS [18] employ a *descriptive* approach to provenance: existing processes' interactions with each other and with the underlying computing environment (filesystem, operating system, etc.) are transparently monitored, collected, and assembled *post hoc* into a provenance graph. Neither ES3 nor PASS require a computation to be expressed in any particular form or language. Both are thus attractive to scientists who do not wish to commit to a specific workflow environment, or who wish to extract provenance information from existing informal workflows (e.g. scripts) without rewriting them.

Relative to ES3, the PASS approach could be considered "strategy 0" (see section 2.1), in that it pushes passive monitoring down into the operating system kernel. PASS can potentially track system events at a much finer granularity than ES3, at the cost of a more complex implementation and the need for system-level privileges.

3. ES3 IMPLEMENTATION OF THE CHALLENGE QUERIES

The workflow representation the challenge script (Figure 2) was assembled *post hoc* by ES3, and retrieved from ES3 as a GraphML [19] document. The workflow diagrams are generated by yWorks' yEd [20] graph editor, using reformatted ES3 GraphML documents as input. (Files are represented as circles and transformations as squares. Process arguments are omitted to minimize visual clutter.)

Note that ES3 correctly recovers the fact that, in our version of ImageMagick, the `convert` command is actually a shell script that invokes a `convertb` binary, and reads a `delegates.mgk` configuration file. (The shell is shown as a sub-worklow, whose input is the `convert` script.)

3.1. Query 1

Simple provenance queries like Query 1 are answered in two steps. First, the object whose provenance is being examined is mapped to an ES3 identifier (i.e., UUID), using an ES3 `lookupFile` request:

```
<ES3Request type="lookupFile">
  <select>
    <file>
      <where>
        <domain name="Local file system">
          <identifier>
            <regex>.*atlas x\.gif</regex>
          </identifier>
        </domain>
      </where>
    </file>
  </select>
</ES3Request>
```

Then, the provenance of the object is retrieved with an ES3 `getLineage` request:

```
<ES3Request type="getLineage">
  <traversal>
    <uuidStart>7b25d64b-abbb-4aef-8042-22487d4a7342</uuidStart>
    <direction>back</direction>
    <granularity>link</granularity>
  </traversal>
  <output>
    <format>graphml</format>
    <formatOptions>nested</formatOptions>
    <detailLevel>brief</detailLevel>
  </output>
</ES3Request>
```

Note that this request specifies a back (i.e., ancestry) provenance trace, with results to be returned in GraphML format.

3.2. Query 2

Query 2 is answered similarly to Query 1, with the addition of a termination condition to the traversal specification in the `getLineage` request:

```
<terminationConditions>
  <transformation>
    <where>
      <name>
        <regex>.*softmean</regex>
      </name>
    </where>
  </transformation>
</terminationConditions>
```

I.e., the query is instructed to terminate when an ancestor transformation named `softmean` is encountered.

3.3. Query 3

Query 3 is answered similarly to Query 1, with the addition of a termination condition to the traversal specification in the `getLineage` request:

```
<terminationConditions>
  <levels>5</levels>
</terminationConditions>
```

Although ES3 records the execution of nested workflows (e.g., processing scripts that execute other scripts), it has no notion of explicit workflow stages. Therefore, we elected to answer Query 3 by terminating the query after five consecutive provenance links were traversed, this being equivalent to Stages 3, 4, and 5 in the Challenge workflow.

3.4. Query 4

ES3 answers Query 4 with a single `getTransformation` request, with the transformation name and arguments constrained by regular expressions:

```
<ES3Request type="getTransformation">
  <select>
    <transformation>
      <where>
        <collection>/provenance_challenge/</collection>
        <name>
          <regex>.*align_warp</regex>
        </name>
        <arguments>
          <regex>.*-m\s+12.*</regex>
        </arguments>
      </where>
    </transformation>
```

```
  </select>
</ES3Request>
```

Note that ES3 only partially succeeds for Query 4, since the XQuery language [21] lacks a day-of-week comparison operator.

3.5. Queries 5, 8, and 9

The version of the Probulator used for the Challenge was unable to examine the contents of the objects referenced in the Challenge workflow, or to acquire annotations from ancillary objects. Therefore, ES3 did not answer Queries 5, 8, or 9. (We discuss this further in Section 4.)

3.6. Query 6

ES3 answers Query 6 in three steps. The first step, a `getTranformation` request identical to Query 4, retrieves the `align_warp` invocations that are potential ancestors of Query 6 result set. The second step is a `getLineage` request that traces the provenance links forward from these potential ancestors, stopping at and returning any `softmean` invocations encountered. The third step, another `getLineage` request, retrieves the file objects that are the immediate descendants of these `softmean` invocations.

3.7. Query 7

To answer query 7, we run a modified version of the Challenge workflow script, whose last 3 lines have been changed as follows:

```
#convert atlas-x.pgm atlas-x.gif
pgmtoppm 0.,0.,0.-1.,1.,1. atlas-x.pgm | pnmtojpeg > atlas-x.jpeg
#convert atlas-y.pgm atlas-y.gif
pgmtoppm 0.,0.,0.-1.,1.,1. atlas-y.pgm | pnmtojpeg > atlas-y.jpeg
#convert atlas-z.pgm atlas-z.gif
pgmtoppm 0.,0.,0.-1.,1.,1. atlas-z.pgm | pnmtojpeg > atlas-z.jpeg
```

Note that we allow the added programs to communicate via pipes (as opposed to intermediate files); this is fully supported by ES3.

Having run both the original and modified scripts, we can retrieve their provenance graphs with `getLineage` requests. We then subject the pair of graphs to a locally-developed, simple graph differencing tool. Differences between the original and modified graphs are flagged on a per-element basis (ignoring UUIDs), by adding a `diff` attribute to each element, whose value may be either true or false. These marked-up graphs are then rendered graphically with their differences highlighted (Figures 3a and 3b.)

Our solution to Query 7, while not implemented entirely as ES3 core queries, is nevertheless responsive to one of the primary classes of user queries that ES3 as whole was designed to support; namely, "what changed?" queries. It is extremely common for scientists developing *ad hoc* workflows to notice differences in outputs across invocations between which nothing was

changed. Our graph-differencing approach is designed to answer the "what changed?" query as directly (and visually) as possible, while still allowing subsequent drill-down into the details.

4. CONCLUSION

The Provenance Challenge highlighted ES3s strengths and weaknesses. On the plus side, ES3 seamlessly instrumented the Challenge workflow and recovered all relevant provenance information, with absolutely no modification to any Challenge code, thereby validating the transparency of the ES3 approach to provenance.

On the minus side, Query 4 exposed ES3s dependence on a strictly-conforming XQuery implementation, and Queries 5, 8, and 9 had to be skipped owing to lack of an annotation facility. Both of these shortcomings will be rectified, by the incorporation of an "Annotator" into the ES3 Probulator, and by the (eventual) addition of extended date-time functions to the ES3 web service layer.

The Annotator is a new ES3 component, not available when the Challenge queries were originally addressed. The Annotator runs in parallel with the Logger, in the same environment as the workflow being monitored, but it does not communicate with the plugins, and therefore is not bound by the Logger's real-time constraints. Instead, the Annotator receives messages from the Logger about workflow items (files and programs) that it may wish to examine. Using configuration rules, the Annotator may decide to (for example) calculate a checksum for a data file, or retrieve a `README` file from the same directory a data file or program resides in. These additional metadata (if any) are written to log files that the Transmitter can then "slipstream" into the ES3 messages it assembles from the Logger files. The Annotator is intended to strike a conservative balance between the completely transparent operation of the Logger and the frequent need to obtain higher- level, user-specified metadata.

Our vision for ES3 extends to a constellation of distributed, cooperating installations, reflecting the institutional distribution of the Earth remote sensing community. The next big challenge for ES3 will be to manage distributed provenance information, so that provenance can be traced across multiple system boundaries.

ACKNOWLEDGEMENTS

We thank Michael Colee for assembling and maintaining our computing environment, Greg Janée for advice and encouragement, Kathy Scheidemen for administrative support, and Haavar Valeur for his work on the probulator prototype.

REFERENCES

1. Frew J, Bose R. Earth System Science Workbench: a data management infrastructure for earth science products. In *SSDBM 2001: Thirteenth International Conference on Scientific and Statistical Database Management,* IEEE: 2001; pp.180-189. DOI: 10.1109/SSDM.2001.938550

2. Dozier J, Painter TH. Multispectral and hyperspectral remote sensing of alpine snow properties. *Annual Review of Earth and Planetary Sciences* 2004; **32**:465-494. DOI: 10.1146/annurev.earth.32.101802.120404
3. Maritorena S, Siegel DA. Consistent merging of satellite ocean color data using a semi-analytical model. *Remote Sensing of Environment* 2005; **94**(4):429-440. DOI: 10.1016/j.rse.2004.08.014
4. Valeur, H. Tracking the lineage of arbitrary processing sequences. Masters thesis, Department of Computer and Information Science, Norwegian University of Science and Technology, Trondheim, July 2005. http://urn.ub.uu.se/resolve?urn=urn:nbn:no:ntnu:diva-1341
5. IDL Data Visualization & Analysis Platform. http://www.ittvis.com/idl [27 November 2006].
6. Moreau L, Ludäscher B, Altintas I, Barga RS, Bowers S, Callahan S, Chin J, Clifford B, Cohen S, Cohen-Boulakia S, Davidson S, Deelman E, Digiampietri L, Foster I, Freire J, Frew J, Futrelle J, Gibson T, Gil Y, Goble C, Golbeck J, Groth P, Holland DA, Jiang S, Kim J, Koop D, Krenek A, McPhillips T, Mehta G, Miles S, Metzger D, Munroe S, Myers J, Plale B, Podhorszki N, Ratnakar V, Santos E, Scheidegger C, Schuchardt K, Seltzer M, Simmhan YL, Silva C, Slaughter P, Stephan E, Stevens R, Turi D, Vo H, Wilde M, Zhao J, Zhao Y. The First Provenance Challenge. *Concurrency and Computation: Practice and Experience* 2007; this issue.
7. Krenek A, Sitera J, Matyska L, Dvorak F, Mulac M, Ruda M, Salvet Z. glite job provenance a job-centric view. *Concurrency and Computation: Practice and Experience* 2007; this issue.
8. Simmhan YL, Plale B, Gannon D. Querying capabilities of the karma provenance framework. *Concurrency and Computation: Practice and Experience* 2007; this issue.
9. Zhao J, Goble C, Stevens R, Turi D. Mining tavernas semantic web of provenance. *Concurrency and Computation: Practice and Experience* 2007; this issue.
10. Futrelle J, Myers J. Tracking provenance semantics in heterogeneous execution systems. *Concurrency and Computation: Practice and Experience* 2007; this issue.
11. Barga RS, Digiampietri LA. Automatic capture and efficient storage of escience experiment provenance. *Concurrency and Computation: Practice and Experience* 2007; this issue.
12. Ludäscher B, Podhorszki N, Altintas I, Bowers S, McPhillips TM. Models of computation and provenance, and the RWS approach. *Concurrency and Computation: Practice and Experience* 2007; this issue.
13. Schuchardt K, Gibson T, Stephan E, Chin G, Applying content management to automated provenance capture. *Concurrency and Computation: Practice and Experience* 2007; this issue.
14. Clifford B, Foster I, Hategan M, Stef-Praun T, Wilde M, Zhao Y. Tracking provenance in a virtual data grid. *Concurrency and Computation: Practice and Experience* 2007; this issue.
15. Scheidegger C, Koop D, Santos E, Vo H, Callahan S, Freire J, Silva C. Tackling the provenance challenge one layer at a time. *Concurrency and Computation: Practice and Experience* 2007; this issue.
16. Kim J, Deelman E, Gil Y, Mehta G, Ratnakar V. Provenance trails in the wings/pegasus system. *Concurrency and Computation: Practice and Experience* 2007; this issue.
17. Cohen-Boulakia S, Biton O, Cohen S, Davidson S. Addressing the provenance challenge using zoom. *Concurrency and Computation: Practice and Experience* 2007; this issue.
18. Seltzer M, Holland DA, Braun U, Muniswamy-Reddy KK. Pass-ing the provenance challenge. *Concurrency and Computation: Practice and Experience* 2007; this issue.
19. Brandes U, Eiglsperger M, Herman I, Himsolt M, Marshall MS. GraphML progress report: structural layer proposal. In *Proc. 9th Intl. Symp. Graph Drawing (GD '01)*, LNCS **2265**:501-512; Springer-Verlag: 2002.
20. yEd - Java^{TM}Graph Editor. http://www.yworks.com/en/products_yed_about.htm [27 November 2006].
21. Malhotra A, Melton J, Walsh N. XQuery 1.0 and XPath 2.0 functions and operators. W3C Proposed Recommendation 21 November 2006. http://www.w3.org/TR/xquery-operators [27 November 2006].