ES3: A Demonstration of Transparent Provenance for Scientific Computation*

James Frew and Peter Slaughter

Donald Bren School of Environmental Science and Management
University of California, Santa Barbara, CA 93106-5131, USA
{frew,peter}@bren.ucsb.edu
http://eil.bren.ucsb.edu

Abstract. The Earth System Science Server (ES3) is a software environment for data-intensive Earth science, with unique capabilities for automatically and transparently capturing and managing the provenance of arbitrary computations. Transparent acquisition avoids the scientist having to express their computations in specific languages or schemas for provenance to be available. ES3 models provenance as relationships between processes and their input and output files. These relationships are captured by monitoring read and write accesses at various levels in the science software and asynchronously converting them to time-ordered streams of provenance events which are stored in an XML database. An ES3 provenance query returns an XML serialization of a provenance graph, forward or backwards from a specified process or file. We demonstrate ES3 provenance by generating complex data products from Earth satellite imagery.

Keywords: ES3; provenance; instrumentation; passive; transparency.

1 Introduction

The Earth System Science Server (ES3) is a software environment for data-intensive Earth science. ES3 has unique capabilities for automatically and transparently capturing, managing, and reconstructing the provenance of arbitrary, unmodified computational sequences [1]. *Automatic* acquisition is critical to avoid the inaccuracies and incompleteness of human-specified provenance (i.e., annotation.) *Transparent* acquisition avoids the computational scientist having to learn, and be constrained by, a specific language or schema in which their problem must be expressed or structured for provenance to be available.

Unlike most other provenance management systems, ES3 captures provenance from running processes, as opposed to extracting it from static specifications such as scripts or workflows. ES3 provenance management can thus be added to any existing scientific computations, without modifying or re-specifying them.

* This work was supported by National Aeronautics and Space Administration cooperative agreements NNG04GC52A and NNG04GE66G.

J. Freire, D. Koop, and L. Moreau (Eds.): IPAW 2008, LNCS 5272, pp. 200–207, 2008.

2 Model and Methodology

ES3 models provenance in terms of processes and their input and output files. We use "process" in the classic sense of a specific execution of a program. In other words, each execution of a program or workflow, or access to a file, yields a new set of provenance events.

Relationships between files and processes are deduced by monitoring read and write accesses. This monitoring can take place at the levels of system calls (using `strace`), library calls (using instrumented versions of application libraries), and arbitrary checkpoints within source code (using automatically invoked source-to-source preprocessors for specific environments such as IDL [2].) Any combination of monitoring levels may be active simultaneously, and all are transparent to the scientist-programmer using the system.

ES3 provenance is the directed graph of files and processes resulting from a specific invocation event (e.g., a "job".) Nested processes (processes that spawn other processes) are correctly represented. In addition to retrieving the entire provenance of a job, ES3 supports arbitrary forward (descendant) and/or reverse (ancestor) provenance retrieval, starting at any specified file or process.

3 Implementation

ES3 is implemented as a provenance-gathering client and a provenance-managing server (Figure 1.) The client runs in the same environment as the processes whose provenance is being tracked.

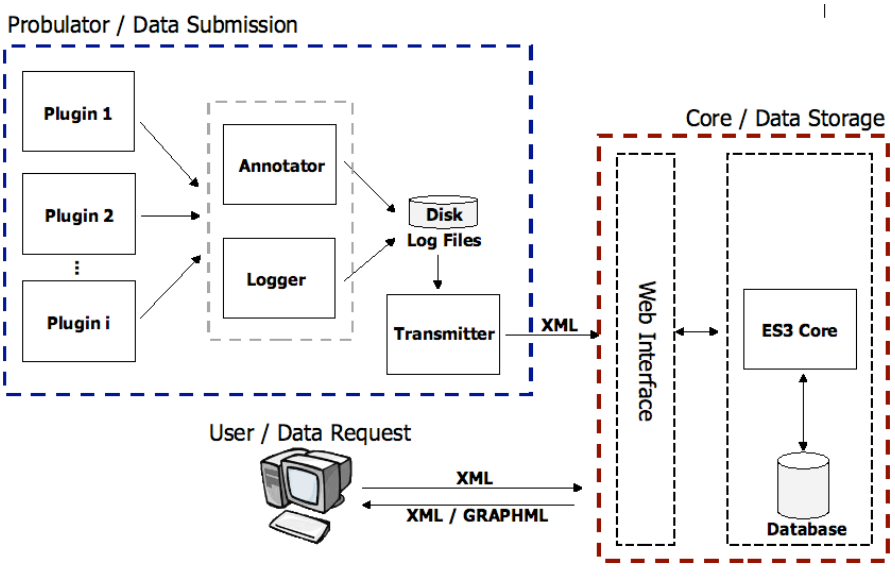

Fig. 1. ES3 architecture

The client is a set of *logger* processes that intercept raw messages from the various monitoring modes (*plugins*) and write them to log files. A separate *annotator* client optionally examines the files and directories being accessed by the instrumented processes and retrieves certain kinds of non-provenance metadata (e.g., `README` files and source code comments.) A common *transmitter* client asynchronously scans the log files, assembles the provenance events into a time-ordered stream, assigns UUIDs to each file and process being tracked, and submits a raw provenance report to the *ES3 core* (server.)

The ES3 core is an XML database with a web service middleware layer that supports insertion of file and provenance metadata, and retrieval of provenance graphs. File metadata allows ES3 to track the one-to-many correspondence between external file identifiers (e.g., pathnames) and internal (UUID) references to those files in provenance reports. Provenance queries cause the ES3 core to assemble a provenance graph (by linking UUIDs) starting at a specified process or file and proceeding in either the ancestor or descendant direction. The graphs are returned serialized in various XML formats (ES3 native, GraphML [3], etc.), and can be rendered visually by tools such as Graphviz [4] and yEd [5].

4 Applications

ES3 is particularly useful for elucidating "hidden" provenance—dependencies between files and processes that aren't explicitly stated in the workflows or scripts that invoke the processes—and for managing highly nested provenance graphs. We give examples of each of these capabilities in this section.

4.1 Hidden Provenance

The final step in the First Provenance Challenge [6] workflow invokes a procedure `convert` that converts images from one format to another (Figure 2.) In the script implementing this workflow, the `convert` operations appear to be atomic commands:

```
convert atlas-x.pgm atlas-x.gif
convert atlas-y.pgm atlas-y.gif
convert atlas-z.pgm atlas-z.gif
```

The ES3 provenance for this portion of the challenge workflow reveals `convert` a more complex picture (Figure 3.) Each invocation of `convert` is actually a shell process which reads the `convert` *script* as input. These processes, correctly depicted as nested workflows, invoke the otherwise hidden command `convertb` with an otherwise hidden input file `delegates.mgk` (a configuration file for the ImageMagick [7] software package.) Workflow-based *a priori* provenance would be unlikely to capture this level of detail.

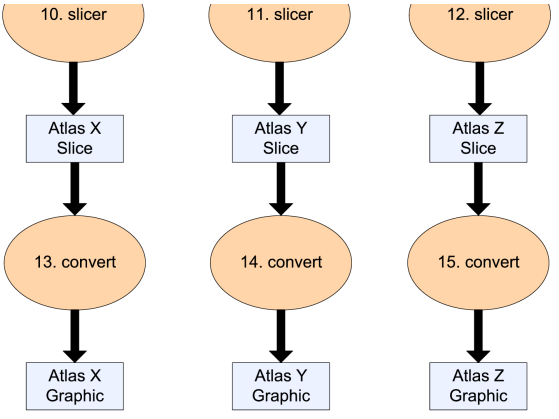

Fig. 2. `convert` operation in challenge workflow

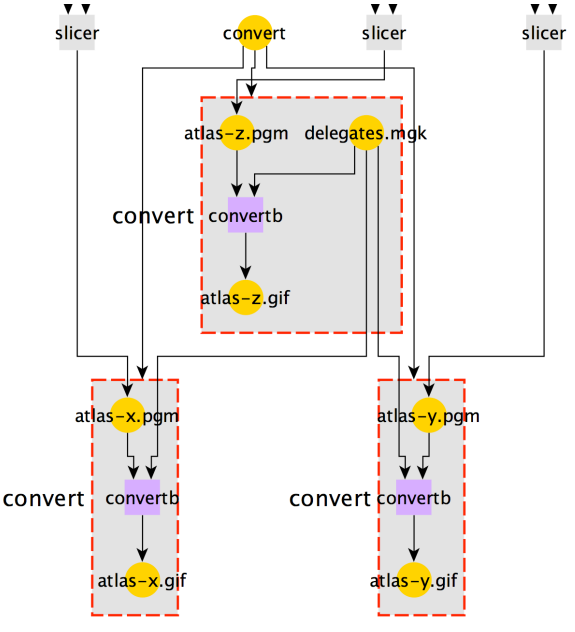

Fig. 3. ES3 provenance for `convert` operation

4.2 Nested Provenance

We use ES3 to track the provenance of a snow-covered-area data product, derived from satellite imagery of portions of the Sierra Nevada (California) mountain range (Figure 4.) The snow product involves processing steps implemented in IDL, C, and UNIX shell scripts, and the algorithms are under active development.

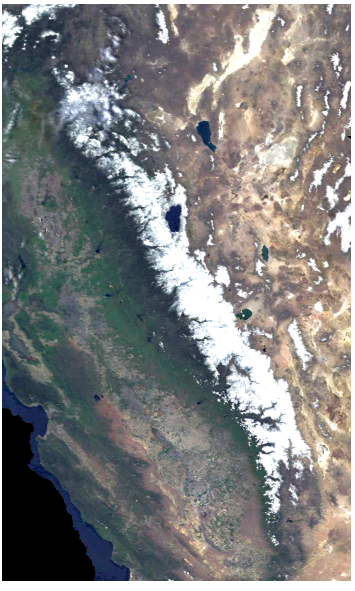

Fig. 4. MODIS satellite image of Sierra Nevada

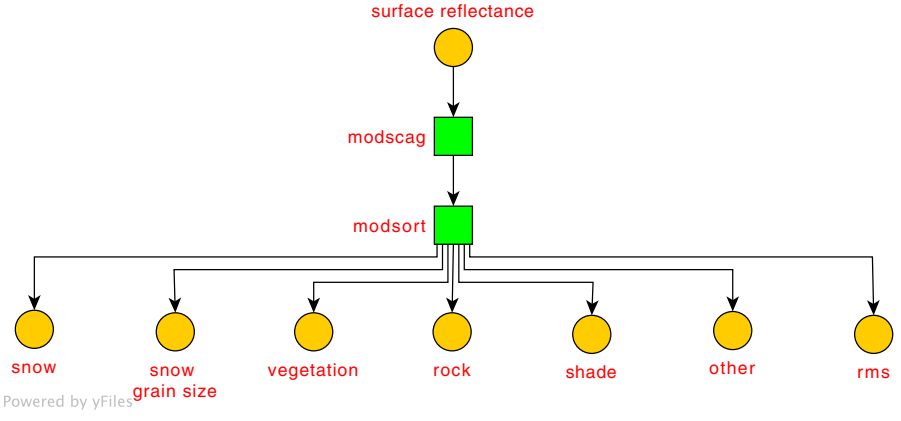

Fig. 5. Snow product top level workflow

Figure 5 shows an idealized top-level workflow for the product. A satellite image of surface reflectance (albedo) is processed by `modscag` into multiple estimates of the surface composition of each pixel. `modsort` select the best of these estimates for each pixel and creates a suite of output grids whose cell values are the percentage of snow (Figure 6) and other components present at the corresponding surface location, as well as estimates of snow grain size, classification error, and whether the input pixel was too deeply shaded by surrounding terrain to be usable.

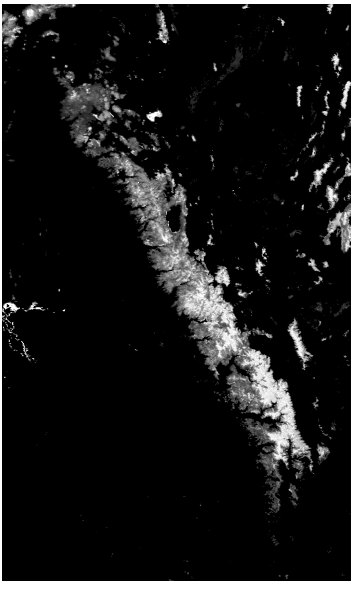

Fig. 6. Fractional snow-covered area, Sierra Nevada

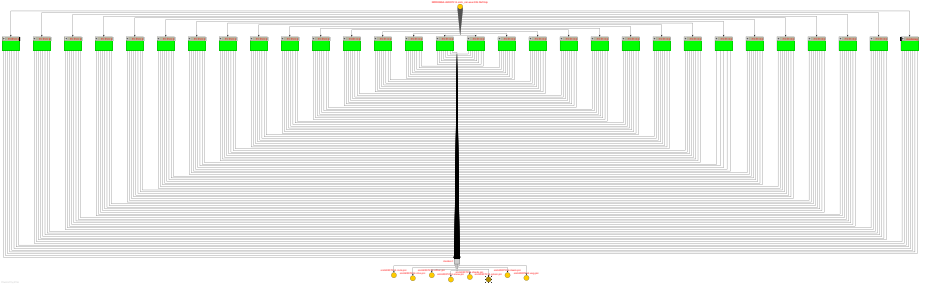

Fig. 7. Snow product provenance (`modscag` nesting expanded)

Requesting forward provenance for an actual satellite image (Figure 7) reveals that the `modscag` workflow step actually comprises 30 separate invocations of the `modscag` program (each of which uses different starting assumptions about surface composition), which `modsort` merges into a single set output files.

The ES3 request that yielded Figure 7 included a restriction to avoid expanding nested workflows. Relaxing this restriction for an entire `modscag` workflow would yield a provenance graph too complex for a printed page. Instead, Figure 8 shows the combined forward and reverse provenance for a single one of the 30 `modscag` program invocations. (Imagine variations on Figure 8 replacing all 30 processes in Figure 7 to get an idea of the complexity of a complete `modscag` "run.")

Note that since Figure 8 is a portion of a much larger provenance graph, it provides sufficient information for some provenance assertions but not others.

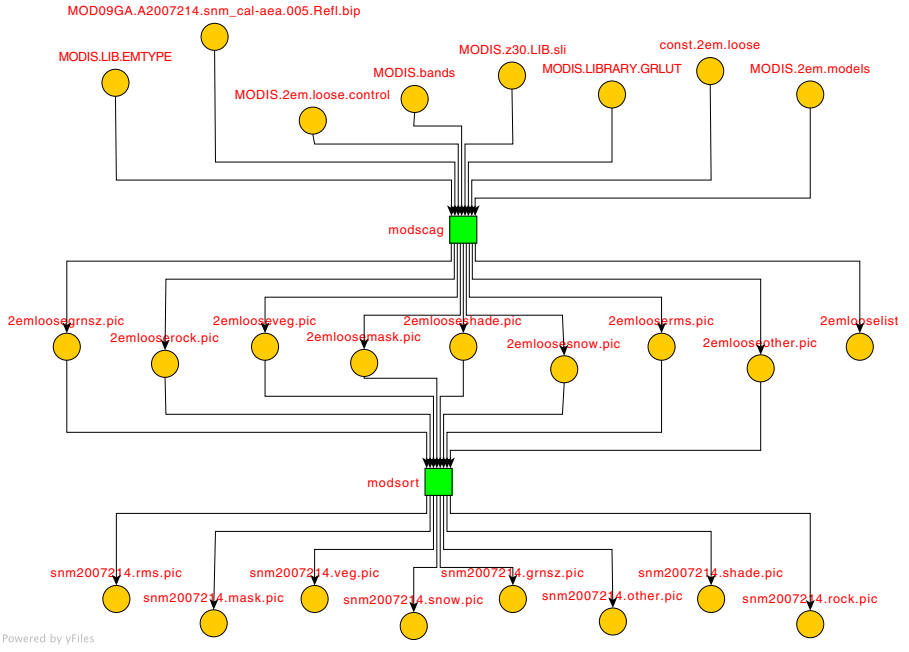

Fig. 8. Snow product provenance (single `modscag` invocation detail)

For example, it correctly shows that the file `snm2007214.snow.pic` is derived from the image `MOD09GA.A2007214.snm_cal-aea.005.Refl.bip`, but does not show any of `snm2007214.snow.pic`'s possible antecedents from any of the other 29 `modscag` invocations.

5 Demonstration

The ES3 client currently includes plugins for the `bash` shell and the IDL interpreted programming language. The ES3 server comprises Java servlets running in Tomcat. The ES3 demo runs on a standalone Linux host (optionally accessing a remote ES3 server) and includes sample shell scripts and IDL programs taken from production science applications. Demo users can run and modify the scripts (including adding and deleting applications), issue arbitrary provenance requests, and graphically explore the resulting provenance graphs and their attached metadata.

Acknowledgments. We thank Michael Colee for assembling and maintaining our computing environment, Greg Janée for advice and encouragement, Dominic Metzger for his work on the probulator, Thomas Painter for supplying the `modscag` test case, and Kathy Scheidemen for administrative support.

References

1. Frew, J., Metzger, D., Slaughter, P.: Automatic capture and reconstruction of computational provenance. Concurrency and Computation: Practice and Experience 20(5), 485–496 (2008)
2. The IDL Computing Environment for Data Visualization & Analysis from ITT, http://www.ittvis.com/idl
3. Brandes, U., Eiglsperger, M., Herman, I., Himsolt, M., Marshall, M.S.: GraphML Progress Report–Structural Layer Proposal. In: Mutzel, P., Jünger, M., Leipert, S. (eds.) GD 2001. LNCS, vol. 2265, p. 501. Springer, Heidelberg (2002)
4. Gansner, E.R., North, S.C.: An open graph visualization system and its applications to software engineering. Software—Practice and Experience 30(11), 1203–1233 (2000)
5. yEd - Java Graph Editor, http://www.yworks.com/products/yed
6. Moreau, L., Ludäscher, B., Altintas, I., Barga, R.S., Bowers, S., Callahan, S., Chin, G., 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, D.A., 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, Y.L., Silva, C., Slaughter, P., Stephan, E., Stevens, R., Turi, D., Vo, H., Wilde, M., Zhao, J., Zhao, Y.: Special Issue: The First Provenance Challenge. Concurrency and Computation: Practice and Experience 20(5), 409–418 (2008)
7. ImageMagick: Convert, Edit, and Compose Images, http://www.imagemagick.org