BURRITO: Wrapping Your Lab Notebook in Computational Infrastructure

BURRITO: Wrapping Your Lab Notebook in Computational Infrastructure

Philip J. Guo
Stanford University

Margo Seltzer
Harvard University

Abstract

Researchers in fields such as bioinformatics, CS, finance, and applied math have trouble managing the numerous code and data files generated by their computational experiments, comparing the results of trials executed with different parameters, and keeping up-to-date notes on what they learned from past successes and failures.

We created a Linux-based system called BURRITO that automates aspects of this tedious experiment organization and notetaking process, thus freeing researchers to focus on more substantive work. BURRITO automatically captures a researcher's computational activities and provides user interfaces to annotate the captured provenance with notes and then make queries such as, *"Which script versions and command-line parameters generated the output graph that this note refers to?"*

1 Motivation

For hundreds of years, the pace of scientific research was relatively slow, limited by the need to set up, run, and debug experiments on physical apparatus. Throughout their experiments, researchers took the time to write meticulous accounts of their hypotheses, observations, analyses, and reflections in handwritten *lab notebooks*.

In the past few decades, the pace of research has sped up significantly as more experiments are being done on the computer. People as diverse as the bioinformatics Ph.D. student who is testing out sequence alignment algorithms, the computer performance engineer who is tuning optimization parameters, and the web marketing analyst who is trying to find the best set of clickstream features to predict purchase rates, all struggle with the same problems in their experimental workflow [4, 6]:

- To test hypotheses, they constantly adjust their code and re-execute to generate numerous output data files. They struggle to remember which exact changes to their code generated a particular output.
- They consult a myriad of resources such as documentation web pages, PDFs of related papers, code snippets, and hand-drawn sketches while they work, so they struggle to remember which resources influenced them to make specific edits to their code.
- They struggle to maintain up-to-date notes on which experimental trials worked and did not work. Since they rapidly edit code, tune execution parameters, and generate new variants of output data, notes taken only a few hours ago might be outdated.

Researchers cope with the above problems by taking notes using a mix of plain-text files, "sticky notes" widgets, and notebook software such as Microsoft OneNote. The fundamental shortcoming of all existing electronic notetaking solutions is that they are not linked with the user's *activity context*, which we define as the user's actions at a particular time, such as editing code, reading documentation, and executing commands. Instead, these tools are simply digital versions of paper lab notebooks. As a result, researchers have trouble organizing their notes and associating them with the proper context. These observations highlight the need for an *electronic lab notebook* that keeps up with the fast pace of computational research rather than mimicking paper notebooks.

2 The BURRITO System

Researchers often work in a heterogeneous environment where they cobble together a patchwork of ad-hoc scripts written in multiple languages, interfacing with a mix of 3rd-party libraries and executables from disparate sources within a command-line environment [6]. We created a Linux-based provenance collection and notetaking system called BURRITO for this target audience. Related work such as scientific workflow systems (e.g., Kepler, Taverna, VisTrails) provide similar provenance collection and annotation features for researchers who work exclusively within those environments [1].

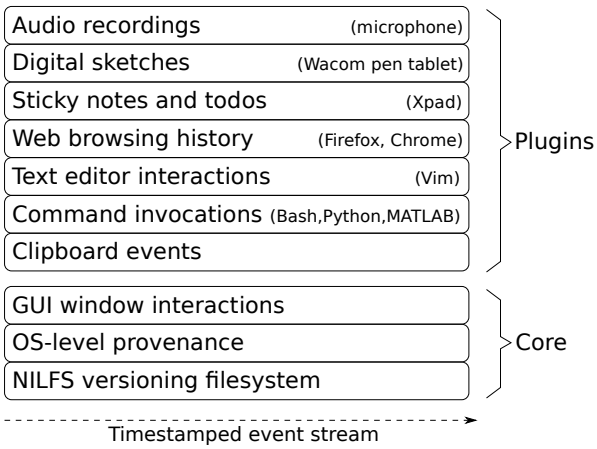

Figure 1: BURRITO platform overview: Each component logs a timestamped stream of events to a master database.

BURRITO consists of two parts: an extensible *platform* that automatically captures provenance and user activity context and a set of *applications* that allow the user to annotate and query the captured metadata stream.

3 BURRITO Platform

The BURRITO platform (Figure 1) consists of a core and a set of plugins that integrate a rich history of user activity into a master MongoDB database on the user's machine. Suppose that Alice is a Ph.D. student conducting computational research on a Linux machine with the BURRITO core platform installed. The NILFS versioning filesystem [2] automatically preserves *all old versions* of her source code and data files, so that she no longer needs to use version control or embed metadata within filenames. An OS-level provenance collection daemon captures execution context, such as which processes read from and wrote to which files, and builds up a provenance graph similar to PASS [5]. A GUI trace daemon captures all of her GUI window interactions, which provides context such as which application windows she is viewing at all times as she is working on her experiments.

BURRITO platform plugins capture Alice's activities within specific applications. For example, the Bash plugin records her executed Bash shell commands, the Firefox plugin records her web browsing history, and the clipboard plugin traces what code she copies and pastes from, say, tutorial websites into her scripts. Alice can also write plugins for scientific software that she uses.

In sum, the BURRITO platform automatically captures Alice's work activities and their context with no perceptible run-time slowdowns and disk space usage of $\sim$2 GB per month (estimated by our own experiences running BURRITO over a two-month period while developing it).

4 BURRITO Applications

We have built four applications atop the BURRITO platform. These applications provide innovative ways of interacting with provenance beyond simply exploring a traditional interactive graph-based visualization [3].

4.1 Activity Feed

The Activity Feed is a sidebar residing on the left portion of the user's Linux desktop background. It periodically polls the master BURRITO database (every 5 seconds by default) and displays a near real-time stream of the user's actions as a list of *feed events* in reverse chronological order. New events appear at the top of the feed and push down older events (Figure 2). This UI metaphor is inspired by the Facebook news feed and Twitter tweet stream. The feed currently displays six types of events:

- A **Bash command event** shows a group of Bash shell commands executed in the same directory without any other intervening events. The user can click on any command to copy it to the clipboard and paste it into a terminal to re-execute.
- A **website visit event** shows a set of web pages visited without any intervening events. The user can click on any page title to open its link in a browser.
- A **file modification event** shows a group of files modified by a particular process. For example, saving a source code file in a text editor will create a new file modification event, as does executing a script to generate an output data file.
- A **digital sketch event** shows a thumbnail view of a sketch that the user has just drawn using, say, a digital pen tablet. Researchers often draw freehand sketches and doodles while they work.
- The user can create a **status update event** by entering text in the status text box and pressing the "Post" button. This is the main way for users to take notes about what they are currently working on at a given moment, which helps place other events in context. (e.g., posting *"I'm now trying to optimize my B-tree split algorithm to copy less data"*).
- The user can create a **checkpoint event** by clicking on either the "Happy Face" or "Sad Face" button and then entering a note in the pop-up text box describing why they are happy or sad about the current state of the experiment. The system takes a screenshot and pushes it alongside the note onto the feed. A "happy checkpoint" is like making a *commit* in a version control system, and a "sad checkpoint" is like filing a *bug report* in a bug tracking system.

Figure 2: The Activity Feed resides on the desktop background and shows a near real-time stream of user actions.

4.1.1 Annotating feed events

Besides writing notes in status update and checkpoint events, the user can also add text annotations to all other types of feed events. After right-clicking on an item in the feed (e.g., a Bash command invocation), the user can choose the "Annotate" option from a pop-up menu. Doing so creates a text box immediately below the event where the user can enter and save a note to the database.

Use case: Researchers struggle with managing notes files. For instance, when reading months-old notes about tweaks made to a particular script, the user will probably be unable to view the exact version of the script to which the notes refer. Activity Feed annotations allow users to make notes within the most precise context *at the time when relevant events are occurring*. The user can later retrieve the exact old version of a file to which an annotation refers, even if that file has since been deleted.

4.1.2 Interacting with file modification events

The Activity Feed provides a convenient interface to monitor and access old versions of files. Right-clicking on a file modification event in the feed pops up a menu where the user can make four types of actions: *1.)* Open the version of the file either right before or after the given modification using a NILFS snapshot. *2.)* Diff two chosen versions of a text file. *3.)* Revert a file to any older version, thus undoing unintended edits. *4.)* View the computational or activity context surrounding modifications to the chosen file (see Sections 4.2 and 4.3).

Use case: The Activity Feed subsumes the basic features of version control systems and sticky notes applets using a unified interface. Rather than restoring files by time or version control commit points, users can access old file versions within the context of all past activities (e.g., executed commands, visited websites, checkpoints, notes).

4.2 Computational Context Viewer

The Computational Context Viewer allows researchers to answer a central question in their workflow: *"What effects did changes in my source code files have on my experiment's output files?"* The user launches this GUI application with the desired output file (and time bound) as an argument. It displays all versions of the chosen output file in reverse chronological order, the parameters of the executed command that created each version, and the *diffs* of all source files that led to the creation of that output file version via the executed command.

For example, Figure 3 shows three variants of an output graph file generated by a Python data analysis script: a line graph, a bar graph, and a bar graph with three crucial bars highlighted in yellow. The leftmost column shows the respective diffs in the script file that led to each change in the output graph file (i.e., the code change responsible for turning the line graph into a bar, and then the change for highlighting the three bars in yellow).

Use case: Using this graphical interface, researchers can answer the question, *"Which version of my source code produced the graph that looked like this?"* The inline diff view allows users to focus on significant changes without needing to learn to use a version control system.

4.3 Activity Context Viewer

The Activity Context Viewer allows researchers to answer the following question: *"What actions influenced me to make these edits to my source code?"* Researchers rarely edit code in isolation; while they work, they often consult documentation web pages, related research papers in the form of PDF or Word documents, and other source code. They sometimes write code comments, notes, or hand-drawn sketches to explain the rationale for their edits, but doing so is a tedious manual process.

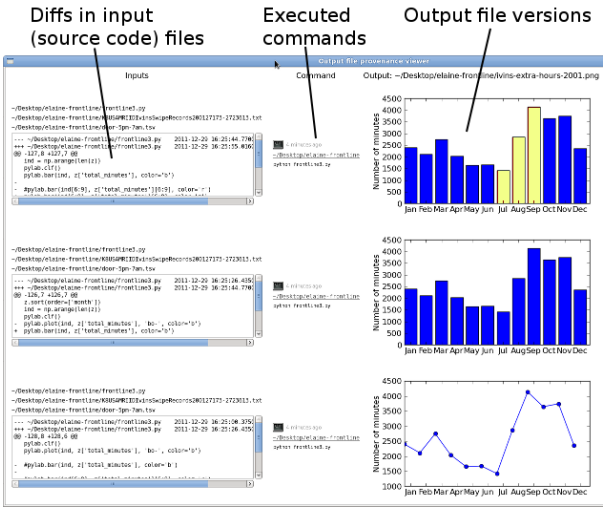

Figure 3: The Computational Context Viewer shows how diffs in input source code files and command-line parameters affect each version of a given output file.

BURRITO automatically captures all of this context surrounding code edits and makes it visible to the user via the Activity Context Viewer. This GUI application looks similar to Figure 3, except that each row displays four fields about one version of the chosen source file:

- **Diffs** of the source file against the previous version.
- **Resources read** while working on edits to the current version, including web pages visited, on-disk documents viewed, and other source code files read.
- **Resources written** while working on this version, including other edited source code files, checkpoints, status updates, and digital sketches drawn.
- **Annotations** that the user can write for this version.

Use case: Researchers can use this tool to re-create their prior work context. For example, *"When I left work last week, I was editing this part of my script and had a collection of reference materials open ... what were they?"*

4.4 Lab Notebook Generator

The Lab Notebook Generator creates an HTML file summarizing the user's activities and notes in a given time period. It currently provides the following functionality: *1.)* Feed events are grouped into *phases* separated by user login sessions, checkpoints, and status updates. *2.)* Within a phase, all files read/written are displayed in a directory tree along with any annotations. Bash commands and visited websites are displayed along with annotations. *3.)* Digital sketches and output image files are rendered as inline HTML images.

Use case: Lab notebook HTML files can be archived, shared with colleagues, and used as the basis for writing status reports and preparing notes for meetings.

5 Future Visions

Although BURRITO can be useful for an individual researcher, we are more excited about discussing the possibility of using it to disseminate knowledge. BURRITO traces could allow students, colleagues, and skeptics to learn from someone's *entire research process*, not just from their final results as presented in a published paper.

In this vision, all Ph.D. students would use BURRITO to capture and archive the complete trials and tribulations of their 5–6 years' worth of experiments. This "Ph.D.-in-a-box" could be used to train new students and to pass down all of the implicit knowledge, experiences, tricks, and wisdom that are rarely captured in a dissertation.

Even more ambitiously, imagine an online library filled with the collected BURRITO archives of all computational research projects. It now becomes possible to perform pattern recognition and aggregation across multiple projects to discover common "tricks-of-the-trade". Someone new to a field, say machine learning, can now learn from the collective "behind-the-scenes" wisdom of thousands of expert machine learning researchers rather than simply reading their published papers.

One could argue that, in the limit, such a system would be like "indexing" all of those researchers' brains and making that knowledge widely accessible. We actually believe that such a system can be more effective than "brain indexing", since people subconsciously apply tricks from their intuitions and often forget the details of what they were working on (especially failed trials). In this vision of the future, a paper is merely a facade for the true contributions of the full research process.

Acknowledgments: Special thanks to Elaine Angelino, Ewen Cheslack-Postava, Eunsuk Kang, Imran Haque, Robert Ikeda, Adam Marcus, Rob Miller, and Jean Yang for giving me helpful feedback on this project and paper.

References

[1] FREIRE, J., KOOP, D., SANTOS, E., AND SILVA, C. T. Provenance for computational tasks: A survey. *Computing in Science and Engineering 10* (May 2008), 11–21.

[2] KONISHI, R., AMAGAI, Y., SATO, K., HIFUMI, H., KIHARA, S., AND MORIAI, S. The Linux implementation of a log-structured file system. *SIGOPS Oper. Syst. Rev. 40* (July 2006).

[3] MACKO, P., AND SELTZER, M. Provenance Map Orbiter: Interactive exploration of large provenance graphs. TaPP '11.

[4] MACLEAN, D. Provenance, PASS & People: A Research Report. Tech. rep., Harvard University, 2007.

[5] MUNISWAMY-REDDY, K.-K., HOLLAND, D. A., BRAUN, U., AND SELTZER, M. Provenance-aware storage systems. USENIX '06, USENIX Association.

[6] PRABHU, P., JABLIN, T. B., RAMAN, A., ZHANG, Y., HUANG, J., KIM, H., JOHNSON, N. P., LIU, F., GHOSH, S., BEARD, S., OH, T., ZOUFALY, M., WALKER, D., AND AUGUST, D. I. A survey of the practice of computational science. In *State of the Practice Reports* (2011), SC '11, ACM.