Provenance Map Orbiter:
Interactive Exploration of Large Provenance Graphs

Peter Macko
Harvard University

Margo Seltzer
Harvard University

Abstract

Provenance systems can produce enormous provenance graphs that can be used for a variety of tasks from determining the inputs to a particular process to debugging entire workflow executions or tracking difficult-to-find dependencies. Visualization can be a useful tool to support such tasks, but graphs of such scale (thousands to millions of nodes) are notoriously difficult to visualize. This paper presents the *Provenance Map Orbiter*, a tool for interactively exploring large provenance graphs using graph summarization and semantic zoom. It presents its users with a high-level abstracted view of the graph and the ability to incrementally drill down to the details.

1 Introduction

Many provenance systems are capable of producing such large amounts of provenance that it is difficult to visualize and analyze. While this is especially true of approaches that operate at the level of an operating system such as PASS [14] or Story Book [18], the problem of scale is more general: For example, the provenance that comes with the MiMI data set (around 6 GB) [12] is significantly larger than the data (270 MB). As systems that capture provenance age, the depths of the provenance graphs continue to grow, and it is reasonable to expect graphs numbering in the millions of nodes. Such graphs capture information about all objects used by a computation, often including information less relevant to the computation itself, e.g., shared libraries. Nonetheless, in some instances, this is precisely the provenance desired to debug configuration problems or software incompatibilities.

Visualization can be a useful tool in supporting such tasks, but large graphs are notoriously difficult to visualize. It seems to be a universal experience that the best graph layouts for provenance are hierarchical, such as *dot* [9] (a part of Graphviz), but their computation usually does not scale beyond a few thousand nodes. The computation of other kinds of layouts, such as force-directed layouts, scale to much larger graphs, but the resulting layouts are too difficult to interpret by end users. Furthermore, it is well-known that node-link diagrams with many thousands of nodes are too large for users to navigate comfortably.

This problem has been historically approached by the provenance community by displaying only the nodes that are "relevant to the user," which are specified using a filter (often expressed as a simple query) [1] or using a view (described using a high-level language, or hard-coded – usually a neighborhood of the node of interest) [16, 19]. We discuss this further in Section 4.

There are many tasks for which the ability to interactively explore the entire provenance graph is beneficial; our goal is to provide that ability without requiring that the user specify queries or views – an error-prone and difficult process for most end users. Debugging workflows, debugging provenance-aware applications, and developing provenance queries are all examples of tasks that be eased through a visualization approach that does not require the user to provide a view or query a priori. We discuss these examples later in Section 2.

The *Provenance Map Orbiter* is a tool for interactively exploring large provenance graphs without requiring prior filtering or view specification. It uses graph summarization to present the user with a high-level view of the provenance graph and semantic zoom [3] to allow her to incrementally explore the details. It combines this ability with the traditional approaches for displaying only "relevant nodes." To date, we have found Orbiter extremely useful in our own provenance analysis, but we would like to solicit input from the community on other ways in which visualization can be generally useful.

In the next section, we make the case for whole-graph exploration. In Section 3, we describe our approach to the problem and relate it to other provenance visualization projects in Section 4. We conclude in Section 5.

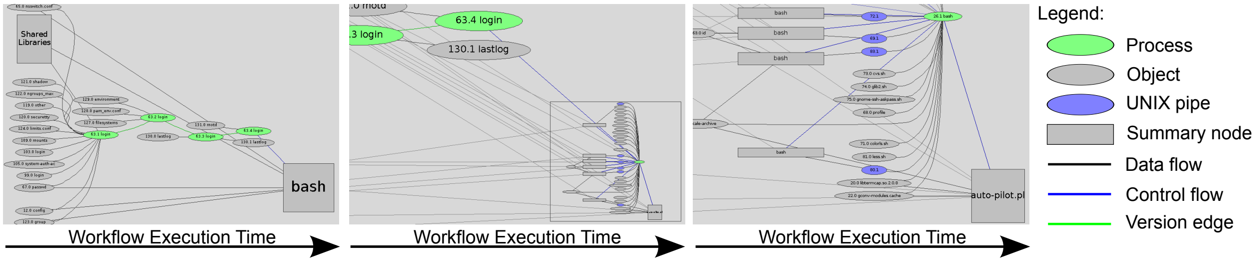


Figure 1: **Semantic Zoom**. Left: a high-level view of Linux's *login* process and its dependencies. Middle & Right: the progress of zooming into the *bash* summary node (in the lower-right corner), representing the login shell and the corresponding user session. Larger versions (with slightly different layout settings) of the first and the third panel are in the appendix.

2 The Case for Whole-Graph Exploration

The traditional approach to provenance visualization shows only the relevant nodes, as specified by a filter, or as defined by a view. While this is sufficient to answer the most common questions about provenance of scientific workflows, the ability to easily explore the entire provenance graph makes it easier for the users to answer many more interesting questions. Most of the scenarios below involve discovering data in the provenance trace that was never supposed to be there. For example:

Debugging Workflows. The user can visually inspect the part of the provenance graph that is relevant to a workflow execution and look for potential problems. Specifying the appropriate filter or a query for this task is fairly tricky, and an incorrect filter might easily cause the user to miss the problem. For example, a typical query asks for all ancestors of the workflow result, but this is not good enough if one of the steps of the computation produced a file that was never supposed to be produced, such as an error log or a core dump. Such a file would not appear in the query result. It is further not uncommon for a workflow result to have tens of thousands of ancestors; thus depending on the scale, the query needs to be even more restrictive to produce a result that is easily visualized. Another problem occurs if the operating system executes a process that does not appear in the lineage, but still affects the computation. For example, starting a substantial *cron* job in the background could influence the results of a benchmark.

Developing Queries. Whole-graph navigation helps in developing complex lineage queries. It rapidly provides the user with a high-level view of what is recorded in the provenance trace. Such views can reveal important details about regions of interest in the graph, such as which node attributes and edge types are relevant for the query. The visualization also helps the developer verify that the query does not omit important details, such as error log files.

Debugging Provenance-Aware Applications. Visualization is useful when integrating provenance from multiple systems (e.g., PASS and StarFlow [2]), as it reveals whether the connections between the two systems are present, correct, etc. We also found visualization to be useful in debugging our various object versioning algorithms in PASS.

Finding Hidden Gems. Finally, while exploring traces collected by the PASS research kernel [14], we found small subgraphs devoted to system-level tasks such as DHCP configuration, setting up SSH connections, or starting a Gnome session. It was interesting to learn, for example, which processes are invoked by *if-up* (the program that brings up a network interface).

3 Description

Orbiter accepts RDF/N3 [4] and OPM [13] data and displays a provenance graph as a node-link diagram, where some nodes correspond to actual objects in the graph, and other nodes correspond to summarizations of collections of nodes from the graph.

3.1 Zoom – Fundamental to Navigation

Semantic zoom [3] is a fundamental part of graph exploration in Orbiter, by which users drill down into summary nodes simply by zooming into them. We built Orbiter around a geographical map metaphor (hence the word "map" in the project's name): The user starts with a coarse-level view of the world and incrementally explores the details by going through different zoom levels, from a view that displays the entire "country" all the way to the "street-level" view.

The process of zoom is illustrated in Figure 1 on an example of a trace in which the user logs into a Linux system and runs a script. The image on the left shows the provenance of Linux's *login* process together with its dependencies and direct descendants. At this level, there are two summary nodes (rectangles in the diagram): a node containing all shared libraries used by *login* (upper left) and a *bash* node containing the interactive login

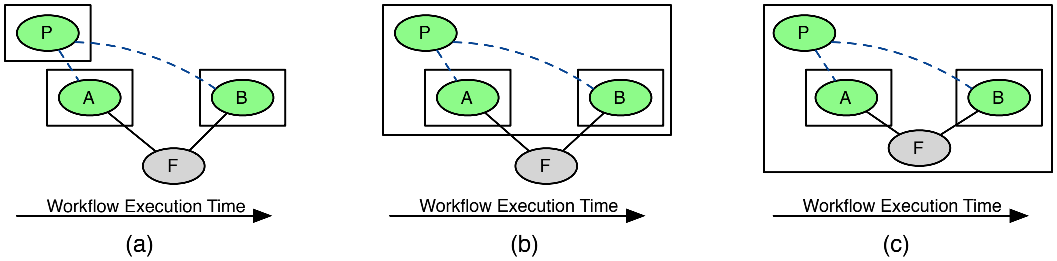


Figure 2: **Graph Summarization**. In this example, process P executed processes A and B. Process A produced file F, which was then read by process B. (a) Each process (primary node) becomes its own summary node (rectangles in the diagram). (b) The summary nodes are placed into a hierarchy using the control flow information. (c) Each secondary node is placed into the summary node that contains all of its ancestors and descendants.

shell and the user's session (lower right).

The following two screenshots show the process of zooming into the *bash* node to reveal more details about the login shell. As the user zooms into the summary node, it expands into multiple nodes showing the additional details, as illustrated by the rightmost screenshot. The nodes show extra information about the dependencies and the child of the actual *bash* process – some of which are again represented as summary nodes. Summary nodes can be nested, so the user can drill into the graph to learn the details in an incremental fashion. We learn that the user executed *auto-pilot.pl* (3rd panel, bottom-right), which is again represented as a summary node. By zooming into this node (figure (c) in the appendix), we learn that the script executed a Blast workload, consisting of *formatdb* and *blastall* processes, and that the results were further processed using a Perl script.

3.1.1 Creating Summary Nodes

Orbiter creates summary nodes as follows: It treats processes as *primary* nodes by default and constructs a summary node for each primary node (process). This is illustrated in Figure 2, part (a), in which the process nodes A, B, and P were placed into their own summary nodes (drawn as rectangles). The designation of primary nodes is configurable, so for example, a user could designate important files as primary nodes.

Orbiter then arranges the summary nodes into a hierarchy. For example, if the primary nodes are processes, they are organized using the process tree reconstructed from the control flow information contained in the provenance trace. For example, in Figure 2, there are control dependency edges denoting that processes A and B were executed by process P, so we move the summary nodes corresponding to A and B inside the summary node for process P. This step is illustrated as part (b) of the figure.

We found that organizing process summary nodes using a process tree produces an easy-to-understand visualization, since processes often correspond to tasks meaningful to the user. They also nest naturally: For example, a *make* process is meaningful to the user, and if she needs more detail, she can zoom into it to see the *gcc* and *ld* processes that constitute the build.

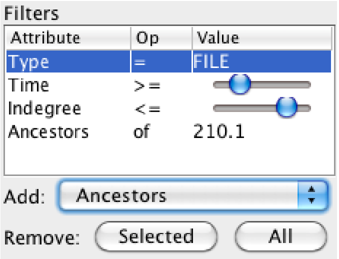
In the next step, illustrated in part (c) of the figure, Orbiter moves all secondary nodes to their appropriate summary nodes. For each secondary node *n*, it obtains a list of its immediate ancestors and descendants, and finds a summary node that contains all of them. It then moves *n* inside that summary node. In our example, the neighbors of file F are processes A and B, which are both contained in P's summary node. Orbiter thus moves F inside this summary node. This step is repeated until no more changes can be done.

The summarization algorithm finishes by combining objects of similar type into the appropriate summary nodes. For example, Orbiter merges all "*.so" files within each summary node generated in the previous steps into a "Shared Libraries" node.

In summary, to create a process summarization, Orbiter creates a summary node for each process (primary node), and then arranges these summary nodes in a way that reflects the process tree reconstructed from the control flow information found in the provenance trace. It then moves all non-processes (secondary nodes) in the appropriate summary nodes. The algorithm is not specific to processes and process trees; it can be applied to any kind of primary node, as long as those nodes can be meaningfully organized into a tree. If the provenance graph contains a collection of objects, they can also be grouped into a summary node.

3.1.2 Advantages and Limitations

Semantic zoom enables Observer to scale to large graphs, since the graph layout can be computed incrementally only for the expanded summary nodes – so that we do not need to compute the layout for the entire graph at once. Using this approach alone, we were able to load a provenance graph with several tens of thousands of nodes



Filters

Attribute	Op	Value
Type	=	FILE
Time	>=	
Indegree	<=	
Ancestors	of	210.1

Add: Ancestors

Figure 3: **Filters & Queries**. The user specifies the filters and simple lineage queries (in the form of special filters) in a filter editor displayed above. The filters are combined with “and.”

into Orbiter and explore it comfortably.

On the other hand, there are cases when the provenance graph does not contain enough semantic information that allows it to be summarized well, or it is a degenerate case – such as when the graph contains nodes with high fanout. Our algorithm might produce several large summary nodes, each of which might be too large to reasonably visualize. In this case, the user can still continue to explore the graph with the aid of filters, queries, and domain-specific views.

Investigating better ways of handling such degenerate cases, as well as finding different methods for graph summarization, is an important part of our future work.

3.2 Filters and Queries

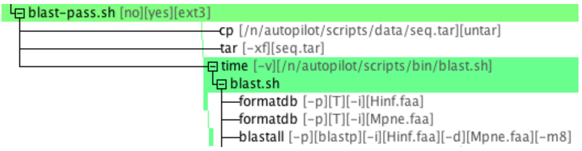
The user can hide unnecessary nodes by specifying filters on the node attributes, such as name, type, or timestamp. The displayed provenance graph updates interactively as the user edits the filters. The filters are applied to the underlying provenance graph, not to the summary nodes.

The user can further combine the filters with simple lineage queries, e.g., find all nodes that are descendants of “auto-pilot.pl” and ancestors of “AB.out”. The queries are specified in the form of filters; the query above would be expressed for node N as: $N \in$ Descendants(“auto-pilot.pl”) $\wedge$ $N \in$ Ancestors(“AB.out”).

The user specifies both the filters and the queries in a filter editor (Figure 3). Orbiter automatically chooses the appropriate editor components based on guidelines inspired by Eisenstein and Puerta [7]. Currently, we specify the nodes in a query by their numerical IDs, but this restriction will be removed in a near future.

3.3 Views

The visualization has limited support for browsing the provenance through views that are not node-link diagrams. For example, we found it useful to implement a view that superimposes the process tree on a timeline (Figure 4). The green segments on the figure correspond to the times the given process was executing, which we found useful for checking whether there was any other process running at the same time as our benchmark. Selecting a process creates an additional filter on the provenance graph, in which only the nodes relevant to the process are displayed.



```
blast-pass.sh [no][yes][ext3]
    cp [/n/autopilot/scripts/data/seq.tar][untar]
    tar [-xf][seq.tar]
    time [-v][/n/autopilot/scripts/bin/blast.sh]
     blast.sh
        formatdb [-p][T][-i][Hinf.faa]
        formatdb [-p][T][-i][Mpne.faa]
        blastall [-p][blastp][-i][Hinf.faa][-d][Mpne.faa][-m8]
```

Figure 4: **Process Tree View**. A view that displays the process tree on the workflow timeline. The green segments correspond to the times the process was running.

The view is currently hard-coded in the visualization application. Enabling the users to specify their own views of the data through a high level language (akin to Adenine [19]) is a part of our future work.

3.4 Other Features

Finally, Orbiter supports other, more standard features, such as: clicking on a node to expose details about the node, search by node attributes, process and file summarization, and version collapsing.

3.5 Implementation

We implemented Orbiter as a Java application. It uses OPM Toolbox [15] and Sesame [17] to parse the OPM [13] and RDF/N3 [4] input formats, respectively. Once the data is loaded, the program uses a custom graph library specifically build for Orbiter, in which summary nodes are first-class citizens. Finally, Orbiter uses Graphviz [10] to compute the layout of nodes within each expanded summary node.

4 Related Work

Several provenance visualization systems provide functionality similar to many of Orbiter’s features. Provenance Browser [1] provides filtering, where filters are expressed as QLP queries. As in Orbiter, the filtered view can be easily incrementally refined. Zoom*UserViews [5] allows the user to select *relevant* nodes, and it then groups irrelevant nodes into summary nodes. But unlike Orbiter, summary nodes cannot nest. Provenance Explorer [11] hides irrelevant nodes behind edges rather than summary nodes. Clicking on an edge shows the hidden nodes, but just like in Zoom*UserViews, this is limited to only a single level of expansion. PLUS [6] allows the users to specify a function that determines an arbitrary representation of a subgraph, but to our knowledge, not recursively.

Nesting of summary nodes is a novel feature in Orbiter, enabling users to incrementally drill down to details. To the best of our knowledge, Orbiter's ability to ingest *and* explore graphs on the order of 10^5 nodes is also unique. Provenance Browser can also ingest large graphs, but the user has to to write a query that returns a small subset to visualize; this might be difficult for some users, and it detracts from the interactive experience.

Finally, there are features we have not yet incorporated into Orbiter. In Haystack [19], users can specify views using a high-level programming language. Probe-It! [16] presents the data in three predefined views, each suitable for a different task. Provenance Browser also supports multiple views, and it has a more general way of querying provenance. We plan to incorporate some of these features, but our emphasis to date has been on supporting interactive viewing of large graphs. We are particularly intrigued by the idea of adapting VisTrails [8] workflow provenance to accelerate the exploration of different filter and visualization settings by tracking the provenance of Orbiter's visualization component.

5 Conclusion

In this paper, we present a technique for interactively exploring large provenance graphs, using graph summarization and semantic zoom to solve the problem of scale. This work is by no means complete: besides the tasks mentioned throughout the paper that need to be tackled, we would like to solicit feedback from the broader provenance community in order to further improve the tool.

6 Acknowledgments

We are grateful to Krzysztof Gajos for suggesting the idea of semantic zoom, and to Uri Braun for ensuring that our implementation of summary nodes follows a sound theoretical model. We thank Hanspeter Pfister for a valuable feedback on an early prototype of Orbiter, and the anonymous TaPP reviewers for their thoughtful comments on this paper.

7 Availability

The copy of the tool can be requested by emailing pass@eecs.harvard.edu.

References

[1] ANAND, M. K., BOWERS, S., ALTINTAS, I., AND LUDSCHER, B. Approaches for exploring and querying scientific workflow provenance graphs. In *IPAW* (2010), vol. 6378 of *LNCS*, Springer, pp. 17–26.

[2] ANGELINO, E., YAMINS, D., AND SELTZER, M. I. StarFlow: A script-centric data analysis environment. In *IPAW* (2010), vol. 6378 of *LNCS*, Springer, pp. 236–250.

[3] BEDERSON, B. B., AND HOLLAN, J. D. Pad++: A zooming graphical interface for exploring alternate interface physics. In *ACM Symposium on User Interface Software and Technology* (1994), pp. 17–26.

[4] BERNERS-LEE, T., AND CONNOLLY, D. Notation3 (N3): A readable RDF syntax. Tech. rep., W3C, 1 2008.

[5] BITON, O., BOULAKIA, S. C., DAVIDSON, S. B., AND HARA, C. S. Querying and managing provenance through user views in scientific workflows. In *ICDE* (2008), IEEE, pp. 1072–1081.

[6] CHAPMAN, A., ALLEN, M. D., BLAUSTEIN, B., SELIGMAN, L., WOLF, C., MORSE, M., AND ROSENTHAL, A. PLUS: Provenance for life, the universe and stuff. Tech. rep., MITRE Corporation, 2010.

[7] EISENSTEIN, J., AND PUERTA, A. R. Adaptation in automated user-interface design. In *International Conference on Intelligent User Interfaces* (2000), pp. 74–81.

[8] FREIRE, J., SILVA, C. T., CALLAHAN, S. P., SANTOS, E., SCHEIDEGGER, C. E., AND VO, H. T. Managing rapidly-evolving scientific workflows. In *IPAW* (2006), vol. 4145 of *LNCS*, Springer, pp. 10–18.

[9] GANSNER, E. R., KOUTSOFIOS, E., NORTH, S. C., AND VO, K.-P. A technique for drawing directed graphs. *IEEE Trans. Software Eng. 19*, 3 (1993), 214–230.

[10] Graphviz: graph visualization software. http://www.graphviz.org/.

[11] HUNTER, J., AND CHEUNG, K. Provenance explorer – a graphical interface for constructing scientific publication packages from provenance trails. *Int. J. on Digital Libraries 7*, 1-2 (2007), 99–107.

[12] JAYAPANDIAN, M., ET AL. Michigan molecular interactions (MiMI): putting the jigsaw puzzle together. *Nucleic Acids Research 35*, Database-Issue (2007), 566–571.

[13] MOREAU, L., CLIFFORD, B., FREIRE, J., FUTRELLE, J., GIL, Y., GROTH, P. T., KWASNIKOWSKA, N., MILES, S., MISSIER, P., MYERS, J., PLALE, B., SIMMHAN, Y., STEPHAN, E. G., AND DEN BUSSCHE, J. V. The open provenance model core specification (v1.1). *Future Generation Comp. Syst. 27*, 6 (2011), 743–756.

[14] MUNISWAMY-REDDY, K.-K., BRAUN, U., HOLLAND, D. A., MACKO, P., MACLEAN, D., MARGO, D., SELTZER, M., AND SMOGOR, R. Layering in provenance systems. In *Proceedings of the 2009 USENIX Annual Technical Conference* (June 2009).

[15] OPM toolbox. http://www.openprovenance.org/toolbox.html.

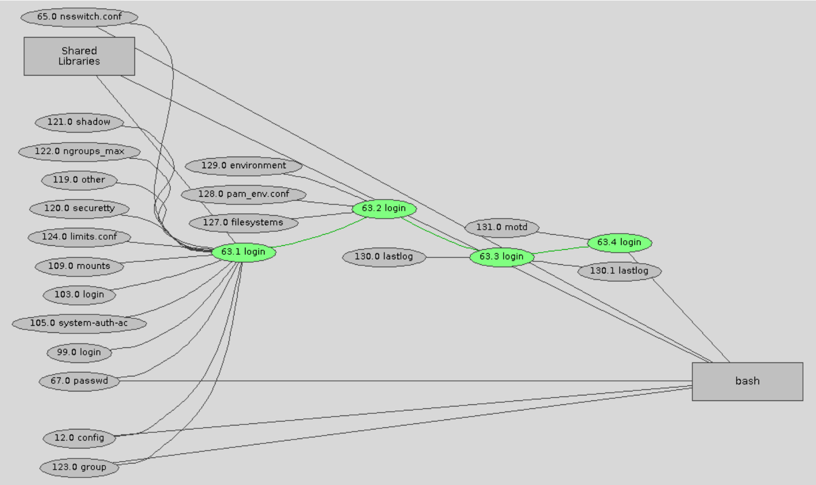
[16] RIO, N. D., AND DA SILVA, P. P. Probe-It! visualization support for provenance. In *Advances in Visual Computing, ISVC* (2007), vol. 4842 of *LNCS*, Springer, pp. 732–741.

[17] Sesame: RDF schema querying and storage. http://www.openrdf.org/.

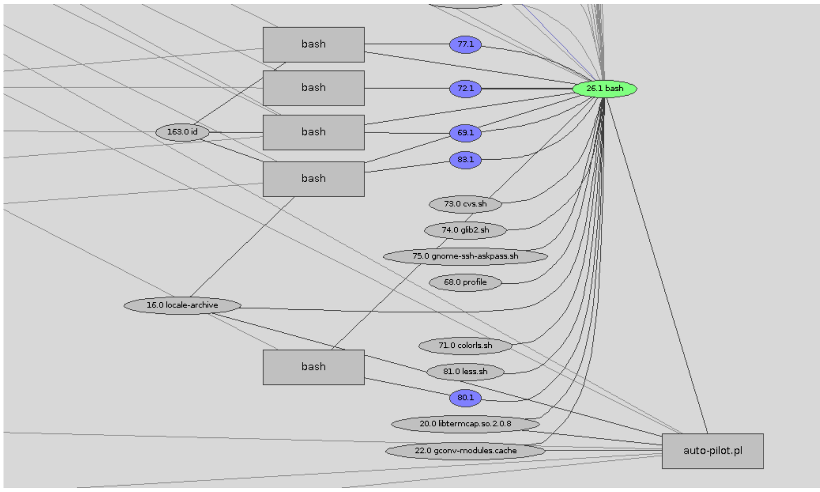
[18] SPILLANE, R. P., SEARS, R., YALAMANCHILI, C., GAIKWAD, S., CHINNI, M., AND ZADOK, E. Story book: An efficient extensible provenance framework. In *TaPP* (2009).

[19] ZHAO, J., WROE, C., GOBLE, C. A., STEVENS, R., QUAN, D., AND GREENWOOD, R. M. Using semantic web technologies for representing e-science provenance. In *International Semantic Web Conference* (2004), pp. 92–106.

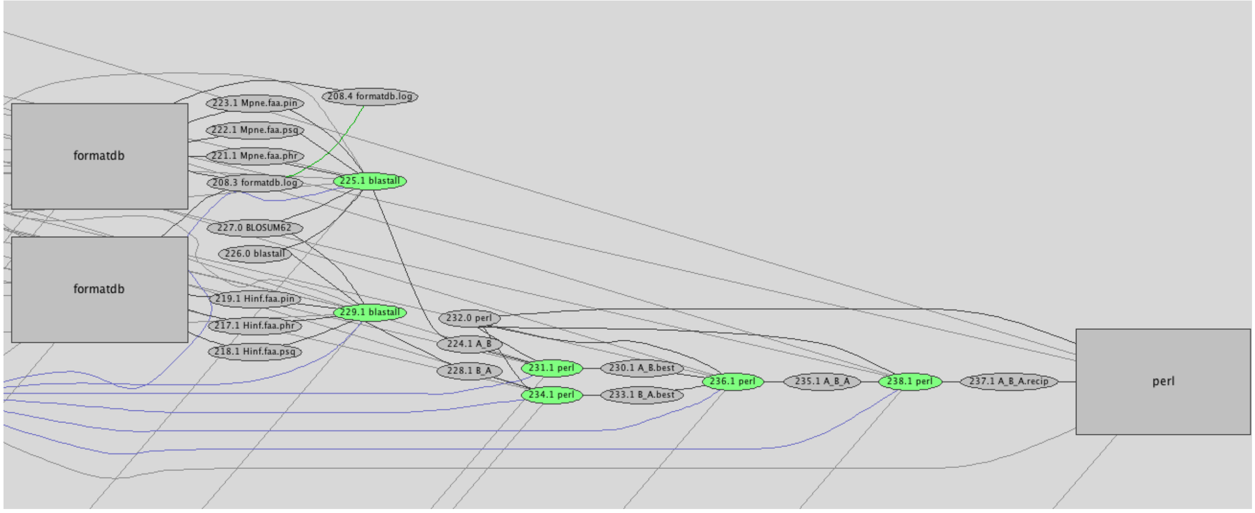
A Screenshots



(a) Larger version of the first panel from Figure 1: Provenance of the *login* process in Linux.



(b) Larger version of the third panel from Figure 1: Provenance of the *bash* login shell in the trace.



(c) A high-level view of the Blast workload contained inside the *auto-pilot.pl* node in the previous picture.