# URSPRUNG: Provenance for Large-Scale Analytics Environments

Lukas Rupprecht
IBM Research–Almaden
Lukas.Rupprecht@ibm.com

James C. Davis*
Virginia Tech, IBM Systems
davisjam@vt.edu

Constantine Arnold
IBM Research–Almaden
Constantine.Arnold@ibm.com

Alexander Lubbock
Vanderbilt University
alex.lubbock@vanderbilt.edu

Darren Tyson
Vanderbilt University
darren.tyson@vanderbilt.edu

Deepavali Bhagwat
IBM Research–Almaden
deepavali.bhagwat@us.ibm.com

## ABSTRACT

Modern analytics has produced wonders, but reproducing and verifying these wonders is difficult. Data provenance helps to solve this problem by collecting information on how data is created and accessed. Although provenance collection techniques have been used successfully on a smaller scale, tracking provenance in large-scale analytics environments is challenging due to the scale of provenance generated and the heterogeneous domains. Without provenance, analysts struggle to keep track of and reproduce their analyses.

We demonstrate URSPRUNG[1], a provenance collection system specifically targeted at such environments. URSPRUNG transparently collects the minimal set of system-level provenance required to track the relationships between data and processes. To collect domain specific provenance, URSPRUNG enables users to specify capture rules to curate application-specific logs, intermediate results etc. To reduce storage overhead and accelerate queries, it uses *event hierarchies* to synthesize raw provenance into compact summaries.

**ACM Reference Format:**
Lukas Rupprecht, James C. Davis, Constantine Arnold, Alexander Lubbock, Darren Tyson, and Deepavali Bhagwat. 2019. URSPRUNG: Provenance for Large-Scale Analytics Environments. In *2019 International Conference on Management of Data (SIGMOD '19), June 30-July 5, 2019, Amsterdam, Netherlands.* ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/3299869.3320235

*Work done while visiting IBM Research–Almaden.
[1]"Ursprung" means "origin" in both German and Swedish.

*SIGMOD '19, June 30-July 5, 2019, Amsterdam, Netherlands*
© 2019 Association for Computing Machinery.
ACM ISBN 978-1-4503-5643-5/19/06…$15.00
https://doi.org/10.1145/3299869.3320235

## 1 INTRODUCTION

Modern data science pipelines have many stages, including data preparation and cleaning, integration, and model selection and tuning [9]. The process is rarely structured and involves many ad hoc iterations over the same data using a multitude of tools and frameworks [11, 15]. This unstructured approach, combined with the rapid increase in generated data [8], makes it challenging for analysts to reproduce, understand, and compare results, leading to duplicated work and limited productivity.

*Provenance* can help solve this problem. It describes the lineage of a data object, e.g., a file, including its origin, the transformations that led to its current state, and the reasons for applying those transformations [5]. The lineage of all data objects forms the *provenance graph*, which can be queried to trace the exact steps necessary for the generation of specific data objects. This not only permits the reproduction of data, but also helps analysts understand its semantics and compare it to other data produced by the same or similar pipelines.

While provenance prototypes have been proposed for over 20 years [16], implementing provenance collection in practice has proved challenging. Provenance collection faces three known problems: (1) the sheer volume of provenance data required to generate a comprehensive provenance graph, incurring expenses during capture, storage, and querying; (2) integration of provenance capture with existing systems, due to the challenge of modifying applications to emit semantic information; and (3) identification of provenance sources and tailoring collection to only what is needed, usually requiring expert knowledge of the entire system.

The problems of provenance collection are exacerbated in analytics environments due to their scale and the variety of applications and pipelines. As a result, existing systems cannot be applied to this case in a straightforward way. They either collect provenance at a too fine-grained granularity [13, 14], require changes to existing applications to support provenance capture [10, 12], or do not capture all relevant provenance [7]. While complete systems exist [6], we believe they are difficult to configure as they require low-level knowledge, e.g., on application function calls.

We showcase Ursprung, a system for collecting provenance in large-scale analytics environments. Ursprung tackles the above three problems with three core design decisions: event hierarchies, application-specific capture rules, and provenance classes. To the best of our knowledge, Ursprung is the first system to combines these techniques to achieve practical provenance collection and querying.

To reduce the amount of captured provenance data, Ursprung uses **event hierarchies**. Event hierarchies synthesize fine-grained system events, such as system calls or individual log entries, into coarser, semantically richer provenance events. For example, a sequence of `fork`, `execve`, and `exit` system calls can be combined into a *process* event with start and finish time, executable, and parameters. By curating the collection of events, Ursprung is able to eliminate noise and produce semantically rich and intuitive provenance.

To collect relevant application-specific provenance, Ursprung relies on the observation that most production software has existing monitoring mechanisms available, e.g., log files or application databases. Rather than instrumenting their toolchains, Ursprung offers a domain-specific language for users to define **application-specific capture rules** to extract provenance from these existing monitoring sources. This declarative way of configuring provenance collection allows to deploy Ursprung in a variety of analytical domains without changes to existing applications.

To tailor the collected provenance to users' needs, Ursprung offers **provenance classes**. Provenance classes are high-level abstractions, which model specific provenance dependencies, e.g., read/write or inter-process communication dependencies. Users select the provenance class(es) of interest, and Ursprung responds by capturing and displaying only the relevant information while filtering all other events. Provenance classes allow users like auditors to configure Ursprung without needing deep system knowledge.

## 2 URSPRUNG OVERVIEW

In this section, we discuss Ursprung's architecture (§2.1) and provide more details on provenance classes (§2.2), event hierarchies (§2.3), and application-specific provenance (§2.4).

### 2.1 Ursprung Architecture

Ursprung follows an event-driven architecture (see Figure 1). It listens for events from core system components, e.g., system calls from the OS or file interactions from the file system, emitted through mechanisms such as Linux's auditd and inotify, or IBM Spectrum Scale's watch folders [3]. Users select different provenance classes to configure, which events should be tracked and which should be filtered.

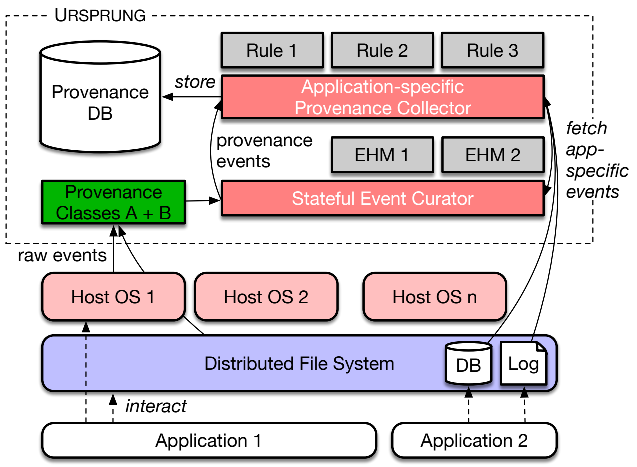


**Figure 1: Ursprung Architecture. An application triggers raw events from sources.**

After filtering, events are passed to the *Stateful Event Curator* (SEC). The SEC uses *Event Hierarchy Models* (EHM) to implement different event hierarchies and produce *provenance events* (see §2.3). Some raw-to-provenance transformations are one to one, while others require to keep state on previous events as the EHM condenses multiple raw events into fewer corresponding provenance events.

After the raw events have been curated, they are passed to the *Application-specific Provenance Collector* (APC). The APC runs the rule engine and evaluates each incoming event against the rules (see §2.4). If a match is found, it executes the rule to extract relevant provenance records from application-specific sources. Any newly extracted raw event is fed back to the SEC for curation, while the original event is stored in the provenance DB. The combination of automatic low-level and rule-based application-specific provenance capture allows Ursprung to support a variety of analytics.

***Prototype Implementation.*** We have built an Ursprung prototype and collected deployment experience as part of a collaboration with Vanderbilt University. The workload is an image processing pipeline consisting of a combination of R and python scripts, built using the Common Workflow Language (CWL) [1], and scheduled via Spectrum LSF [2]. Ursprung is able to collect both low-level process information as well as higher-level data such as CWL workflow statuses. Besides the Vanderbilt workload, we have also tracked provenance for command-line based analyses and Spark jobs.

### 2.2 Provenance Classes

Ursprung categorizes system-level provenance into different provenance classes. Each class is a high-level abstraction, e.g., file-process interactions, and underneath, defines the set of system calls that must be monitored to correctly capture the required provenance. Other examples for classes are inter-process communication or network communication, which require to track additional system calls such as `pipe`, `dup`, `socket`, or `accept`. Presenting system-level provenance in higher-level classes makes it easy for non-expert users to define what should be collected and control the trade-off of overhead vs. collection granularity.

## 2.3 Event Hierarchies

Event hierarchies are used to model provenance entities from raw events. Here, we describe how Ursprung captures the most fundamental provenance entities: processes and files.

***Processes.*** A Process EHM is used to assemble *process provenance events*. Therefor, Ursprung collects raw events for `fork`, `execve`, and `exit` system calls[2] using auditd. A process provenance event includes information like the process's pid (retrieved from `fork`), command line (`execve`), start time (`fork`), and finish time (`exit`). The Process EHM accumulates raw events for each of these system calls, and emits a complete process event once the process' `exit` is observed.

***Files.*** For file-related information, Ursprung uses events provided by IBM Spectrum Scale's watch folder feature. Watch folder events are triggered on file operations such as `rename` or `close` and contain information including path and corresponding pid. Additionally, `close` events contain the number of bytes read from or written to the closed file. Hence, using watch folder events, Ursprung can reliably determine the relationship of the corresponding process to the file without tracking an abundance of `read` and `write` calls, i.e. no event curation is necessary and the File EHM is the identity.

Other storage systems such as Ceph or HDFS provide similar notification mechanisms, which can be supported by Ursprung by implementing the corresponding File EHMs.

## 2.4 Application-specific Provenance

Ursprung captures application-specific provenance to augment the provenance graph. For an analytics job, a user might be interested in both process-related and job-level provenance such as the job's input parameters. Ursprung can retrieve additional provenance data from existing sources, e.g., application logs or databases, by reading from or querying those sources. Therefore, it seamlessly integrates with different applications without having to change them.

Users define capture rules for application-specific sources using Ursprung's domain-specific rule language. A rule consists of a set of conditions on the fields of the incoming provenance event, and corresponding actions. Ursprung implements three actions: `dbload`, `logtransfer`, and `dbtransfer`. `dbload` is used to bulk load a file into the provenance DB

[2]It also includes related calls such as `clone`, `vfork`, and `exit_group`.

```
path=/gpfs/spark/logs/*Master* AND written>0 AND event=CLOSE
->
LOGTRANSFER path MATCH 'Registered app'
FIELDS 1,8,11 DELIM ' '
INTO dbUser:dbPassword@dbHost:dbPort/workflows
USING starttime,name,id
```

**Listing 1: A `logtransfer` rule to extract job information from the Spark master log.**

while `logtransfer` is used to monitor log files for specific entries. `dbtransfer` loads specific rows from a database.

Listing 1 demonstrates a rule to extract the starttime, name, and id of a Spark job. The first part of the rule specifies the conditions: the rule fires if the Spark master log is updated. The second part is a `logtransfer` action, which scans the log file for lines matching `Registered app`, extracts the space-delimited columns 1, 8, and 11 from each matching line, and imports them into the `workflows` table in the database.

## 3 DEMONSTRATION

We now introduce Ursprung's GUI (§3.1) and explain how users can interact with it during the demonstration (§3.2).

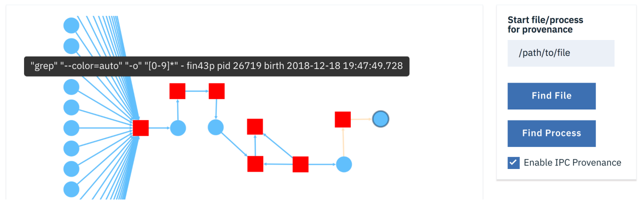
## 3.1 GUI Design

Ursprung's GUI is designed to efficiently traverse and view the provenance graphs. Therefore, it implements an interactive *step-by-step traversal* approach. Users choose initial node(s) to start from and then gradually explore the provenance graph by expanding the neighborhood of a selected node. By limiting expansion to a node's immediate neighbors at a time, users can pace exploration and easier navigate large graphs. The Ursprung prototype uses a relational store for provenance data with a schema optimized for the step-by-step exploration to avoid expensive joins and recursion. However, Ursprung can support other schemas such as PROV-DM [4] by adapting the SEC (see §2).

Ursprung's GUI is implemented as a web application and consists of three parts: a search panel, the provenance graph explorer, and a job display panel.

***Search Panel.*** The search panel (see Figure 2) allows users to search for specific nodes/data objects in the provenance graph. A node can be either a file or a process and the search results form the starting point for exploring the provenance graph. The search panel also has a checkbox to toggle, whether IPC provenance should be displayed or not, as an example for a provenance class.

***Provenance Graph Explorer.*** The explorer is the main GUI component and is designed for interactively traversing the provenance graph (see Figure 2). After selecting an initial node from which to start the exploration (via the search



"grep" "--color=auto" "-o" "[0-9]*" - fin43p pid 26719 birth 2018-12-18 19:47:49.728

/path/to/file

✓ Enable IPC Provenance

**Figure 2: Provenance graph display in Ursprung.**

panel), users can view the dependencies of the node by clicking on it. A click will expand the graph to the neighbors of the clicked node. For example, clicking on a file node will show the processes that wrote to the file to its left and the processes that read the file to its right. Users can view the details of a node, such as the file path, by hovering over it.

***Job Display Panel.*** Ursprung also comes with a panel to display analytics jobs that were run on the cluster. This information is captured from applications using the above presented rule language. This panel currently supports Spark jobs and CWL workflows. Users are able to select a job and view its output files in the graph explorer to start analyzing job provenance. The workflow panel is an example to demonstrate the benefit of application-specific provenance.

## 3.2 Scenarios

We propose three main scenarios. Our goal is to demonstrate how provenance is captured in Ursprung and how it can be used to navigate data in analytics environments.

***Discovery.*** In the Discovery scenario, attendees can explore the provenance of a file to identify its semantics and importance. A user will run forensics on a file of indeterminate origin. Using Ursprung's GUI, the user can search for the file in question and start tracing its dependencies. By analyzing the steps and processes involved in creating the file, the meaning of its content will become clear once traced to its origin and a decision can be made on whether the data is important or can be discarded/moved to cold storage.

***Application-specific Provenance.*** In the second scenario, we demonstrate Ursprung's application-specific provenance capture and its three main rules. We run the Vanderbilt image analysis workload with CWL and Spectrum LSF. Using `dbtransfer` and `logtransfer` rules, we collect CWL- and LSF-related provenance to display the workload in the job display panel. From the panel, attendees can select the workload and view its output files and their provenance. We also show how to use `dbload` rules to collect and index temporary data that is produced as part of the image analysis.

***Interactive.*** The third scenario allows attendees to freely interact with the system via a command line. They can create and delete files and process them using both command line tools and Spark. Attendees can then look at the generated provenance of their processing in real-time. They can also explore, how previous attendees interacted with the system by examining the provenance of existing files.

## 4 RELATED WORK

Provenance collection has received considerable attention in past research. PASS [13] was one of the first systems to provide transparent, system-level provenance capture. Later systems such as PASSv2 [12] and CPL [10] extended the PASS model to also capture application-specific provenance. However, this requires changes to the existing applications to disclose their provenance. Ghoshal et al. introduce the idea of capturing provenance from log files [7] and use a domain-specific rule language to configure the capture. While this is similar to Ursprung, it does not deal with lower-level system provenance. SPADE [6] and more recently CamFlow [14] are complete systems that support both system-level and application-specific provenance without requiring changes to existing applications. However, they do not provide rule-based configuration and provenance classes which makes them harder to configure.

## REFERENCES

[1] 2018. Common Workflow Language. https://www.commonwl.org/.

[2] 2018. IBM Spectrum LSF. https://ibm.co/2Lpafez.

[3] 2018. Introduction to Watch Folder. https://ibm.co/2Fze0Ox.

[4] 2018. PROV-DM: The PROV Data Model. https://bit.ly/2I9TyUN.

[5] Lucian Carata, Sherif Akoush, Nikilesh Balakrishnan, Thomas Bytheway, Ripduman Sohan, Margo Seltzer, and Andy Hopper. 2014. A Primer on Provenance. *Commun. ACM* 57, 5 (2014).

[6] Ashish Gehani and Dawood Tariq. 2012. SPADE: Support for Provenance Auditing in Distributed Environments. In *Middleware'12*.

[7] Devarshi Ghoshal and Beth Plale. 2013. Provenance from Log Files: A BigData Problem. In *EDBT/ICDT Workshops'13*.

[8] Alon Halevy, Flip Korn, Natalya F Noy, Christopher Olston, Neoklis Polyzotis, Sudip Roy, and Steven Euijong Whang. 2016. Goods: Organizing Google's Datasets. In *SIGMOD'16*.

[9] Arun Kumar, Robert McCann, Jeffrey Naughton, and Jignesh M. Patel. 2016. Model Selection Management Systems: The Next Frontier of Advanced Analytics. *SIGMOD Rec.* 44, 4 (2016).

[10] Peter Macko and Margo Seltzer. 2012. A General-Purpose Provenance Library. In *TaPP'12*.

[11] Hui Miao, Amit Chavan, and Amol Deshpande. 2017. ProvDB: Lifecycle Management of Collaborative Analysis Workflows. In *HILDA'17*.

[12] Kiran-Kumar Muniswamy-Reddy, Uri Braun, David A. Holland, Peter Macko, Diana L MacLean, Daniel W Margo, Margo I Seltzer, and Robin Smogor. 2009. Layering in Provenance Systems. In *ATC'09*.

[13] Kiran-Kumar Muniswamy-Reddy, David A Holland, Uri Braun, and Margo Seltzer. 2006. Provenance-Aware Storage Systems. In *ATC'06*.

[14] Thomas Pasquier, Xueyuan Han, Mark Goldstein, Thomas Moyer, David Eyers, Margo Seltzer, and Jean Bacon. 2017. Practical Whole-system Provenance Capture. In *SoCC'17*.

[15] Sebastian Schelter, Joos-Hendrik Böse, Johannes Kirschnick, Thoralf Klein, and Stephan Seufert. 2017. Automatically Tracking Metadata and Provenance of Machine Learning Experiments. In *ML Systems'17*.

[16] Amin Vahdat and Thomas E Anderson. 1998. Transparent result caching. In *ATC'98*.