Curator: Provenance Management for Modern Distributed Systems*†

Warren Smith
The Weather Company, an IBM Business
Andover, MA, USA

Thomas Moyer
UNC Charlotte
Software and Information Systems
Charlotte, NC, USA

Charles Munson
MIT Lincoln Laboratory
Secure Resilient Systems and Technology
Lexington, MA, USA

Abstract

Data provenance is a valuable tool for protecting and troubleshooting distributed systems. Careful design of the provenance components reduces the impact on the design, implementation, and operation of the distributed system. In this paper, we present Curator, a provenance management toolkit that can be easily integrated with microservice-based systems and other modern distributed systems. This paper describes the design of Curator and discusses how we have used Curator to add provenance to distributed systems. We find that our approach results in no changes to the design of these distributed systems and minimal additional code and dependencies to manage. In addition, Curator uses the same scalable infrastructure as the distributed system and can therefore scale with the distributed system.

ACM Reference Format:
Warren Smith, Thomas Moyer, and Charles Munson. 2018. Curator: Provenance Management for Modern Distributed Systems. In *Proceedings of USENIX Theory and Practice of Provenance (TaPP'18).* ACM, New York, NY, USA, 4 pages.

1 Introduction

Data provenance, the history of data as it moves through and between systems, provides distributed system operators with a potentially rich source of information for a wide-range of uses. Operators can use provenance for troubleshooting [13], auditing [6], and forensic analysis [9]. Existing systems have focused on the collection and usage of provenance data, often with the analysis occurring on the same system that collected the provenance. While this works well for applications running on a single host, it quickly breaks down on distributed systems.

In systems that have considered provenance for distributed systems, the proposed architectures treat provenance as unique from other sources of metadata, such as log and audit data. This requires users of provenance to build entire infrastructures to manage the provenance data, increasing the complexity of the application and system. This increased complexity can also increase the attack surface of the application, negating the security benefits of adding provenance to a system. What is needed is a provenance management system that works in concert with existing infrastructures, and provides lightweight integration into applications.

In this paper, we present Curator, a provenance management toolkit that integrates with existing logging/auditing systems. Additionally, Curator provides a lightweight library to integrate provenance collection into existing applications that minimizes dependencies, reducing the integration complexity for application developers. Curator is able to integrate provenance from multiple levels of abstraction, including applications, infrastructure (databases, processing engines, etc.), and operating systems. The system ensures a consistent encoding of data between provenance sources, allowing consumers of provenance data to reason about system behavior across different levels of the system.

We focus our attention on the integration of Curator into microservice-based systems, where applications consist of small services that coordinate to achieve the goals of the application. Such architectures are popular in today's systems and present challenges when adding provenance.

2 Design

The goals for the design of the Curator toolkit emerged from our experiences adding data provenance to prototype data processing systems.

G1) **Minimally invasive:** Our first goal is to make it easier to create and emit provenance from application services and infrastructure. It was often difficult to add and de-conflict packages used by our previous provenance instrumentation with the package dependencies of the data processing systems we were instrumenting.

G2) **Scalable:** The second goal is to aggregate and store provenance information in a scalable way while re-using the infrastructure deployed by a data processing system. Maintainers of data processing systems are often reluctant to add additional software infrastructure solely for the use of a data provenance subsystem, even when the system generates high volumes of provenance data.

*DISTRIBUTION STATEMENT A. Approved for public release: distribution unlimited.
This material is based upon work supported by the Assistant Secretary of Defense for Research and Engineering under Air Force Contract No. FA8721-05-C-0002 and/or FA8702-15-D-0001. Any opinions, findings, conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the Assistant Secretary of Defense for Research and Engineering.

TaPP 2018, July 11–12, 2018, London, UK.

TaPP'18, July 9–13, 2018, London, England

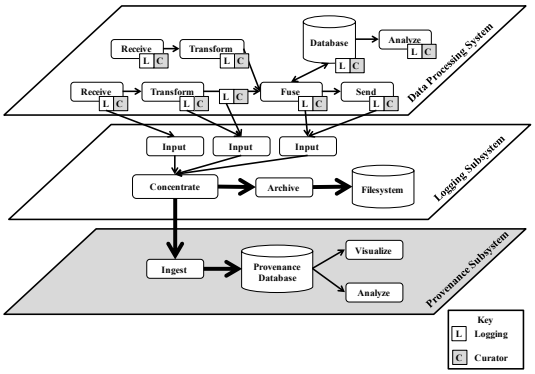

Figure 1. The `Curator` toolkit integrated into a microservice architecture. Some connections have been removed for legibility.

G3) **Visualization:** Our third goal is to provide a set of tools and displays for visualizing and analyzing provenance information. We found that there are commonalities in the provenance visualization and analysis needs of the systems we added data provenance to and that we could create components for use across these systems.

Fig. 1 shows our approach to satisfy these goals. The key concept of this architecture is to view gathering, storing, and analyzing provenance information as a logging problem. The `Curator` design consists of helper libraries to create provenance information and add it to logs, use of a log aggregation and processing system, and a provenance subsystem for storing, visualizing, and analyzing the provenance.

2.1 Instrumentation and Logging

To emit provenance, the `Curator` toolkit includes a Java library to create, encode, and log provenance. Fig. 2 shows the creation of a provenance logger with a serializer, the creation of provenance objects, and the logging of these objects to the same log the microservice uses for its logging.

`Curator` adopts the World Wide Web Consortium's W3C-PROV [5] standards to represent data provenance information and defines a set of Java objects organized as graph vertices and edges to represent the PROV data model. The middle of Fig. 2 contains examples of these objects. To translate provenance objects to and from representations that are suitable for logs, the toolkit contains serializers and deserializers that support formats such as PROV-JSON [8]. The third line of Fig. 2 demonstrates selection of a serializer.

`Curator` makes use of the logging framework chosen by the authors of the microservice, rather than imposing a logging framework, and currently supports log4j [1] and log4j2. The toolkit also provides wrapper `ProvenanceLogger` classes to hold a serializer object and to implement `log()` methods for all of the provenance objects.

2.2 Aggregation and Ingest

Once the microservices log provenance information, the logging system gathers the logs and then filters the provenance information out of the logs. `Curator` deserializes and writes the provenance data into a provenance database. `Curator` includes a simple service to perform these tasks. This service

```
ProvenanceLogger logger =
  new ProvenanceLogger(Logger.getLogger("App"),
                       new ProvJsonSerializer());
Entity input = new Entity();
input.setAttribute("filename", "IMG-0942.jpg");
Activity transform = new Activity();
Used used = new Used(transform, inputData);
logger.log(inputData);
logger.log(transform);
logger.log(used);
```

Figure 2. Adding provenance instrumentation to a microservice.

currently receives log4j data via a socket or a log file. It then deserializes the provenance data using any of the available deserializers (e.g PROV-JSON), and writes the provenance information to any supported provenance database.

However, rather than using the simple `Curator` ingest service, we recommend using a log management system for a more scalable and robust ingest implementation. Systems such as logstash [2] and fluentd [3] are widely used in microservice architectures to manage log data. These services are modular and configurable and support the dual use of monitoring and analyzing the operation of a microservice-based system as well as managing provenance data. The `Curator` toolkit currently includes a logstash output plugin.

2.3 Storage and Query

We have found that it is not possible to select a single data storage solution for provenance data. First, the volume and velocity of provenance data varies from system to system so no one solution is always the most appropriate. Second, a microservice system has likely adopted a database for their needs and the developers and operators would prefer to also use that database for provenance data.

The `Curator` approach is to therefore support a number of different databases behind common `Store` and `Query` interfaces. `Curator` currently supports popular SQL databases (MySQL/MariaDB, PostgreSQL, H2, Derby) and the Accumulo [4] distributed key/value store. `Curator` represents provenance information as graphs of vertices and edges that have attributes (key/value pairs). `Curator` stores these graphs in SQL in a normalized form and in Accumulo in a denormalized form, as is typical for such databases. The optimized schemas used in the databases enable fast retrieval of vertices and edges by their ids and locating vertices and edges that have specific attributes. It has been demonstrated that SQL databases support lower ingest rates and data volumes, but fast queries[5]. Accumulo supports high ingest rates and volumes of data, but queries can be slower [11].

The `Curator Query` interface supports a number of operations to retrieve provenance data to drive analytics. As mentioned above, this interface supports basic operations such as finding vertices and edges by identifier or by attributes. The interface also supports finding ancestors and

[1] https://logging.apache.org/log4j

[2] https://www.elastic.co/products/logstash

[3] https://www.fluentd.org/

[4] https://accumulo.apache.org

[5] This assumes that the tables have the appropriate indexes.

descendants of a vertex (typically an entity) so that an analysis tool can determine what entities, activities, and agents influenced or were influenced by a vertex. For broader views, the Query interface supports finding the ancestors and descendants of a set of vertices and the connected subgraph that a set of vertices are part of.

2.4 Analytics

We use the provenance information available from the Query to drive analytics. For example, we have written system-specific analysis code to analyze the structure and content of a provenance graph related to a data item passing through a data processing pipeline in order to ensure accurate and timely processing of data inputs. This code determines if the provenance graph has the structure and content that we expect and notifies the user if it does not. Such analysis can be used for anomoaly detection and for debugging. However, in the case of anomaly detection, more work is needed to ensure the security of the provenance collected using Curator. Graph-grammar based techniques such as those used in Winnower can be integreated to allow for more general-purpose analysis of graph structure [7].

2.5 Visualization

Finally, provenance visualizations are useful for a number of purposes. An operator or developer can use visualizations to understand the operation of a system and the interactions of the components in the system. They can also use visualizations to analyze the causes or effects of a failure or an attack. Curator includes visualization components for integration into data processing systems. These components provide a general-purpose mechanism to explore provenance data that has been processed by Curator, allowing consumers of provenance data to explore the provenance data in a browser.

3 Use Cases

We used the Curator toolkit to add provenance to several distributed systems. The general purpose of all of these systems is to process data as it streams into the system, store data products, perform analyses on stored data, and present results to users. The systems are architected as a set of microservices that communicate via a message bus.

We integrated Curator into these systems in order to collect provenance. First, we added provenance logging statements into the application services. As described above, these statements are not difficult to write and the application services have a great deal of contextual information that can be of use in the provenance record. However, it is challenging to locate all of the code locations that require logging. One promising technique for auotmatically instrumenting applications for provenance is described in [2]. Such systems can use Curator to add provenance to legacy applications.

Second, we added provenance logging to common application libraries, such as a library that wraps a messaging system to package application data for sending and receiving. This is again easy to accomplish using code similar to that shown in Fig. 2. This approach works well because provenance logging is added in a minimal number of locations and can gather a large amount of the provenance needed for a system. In addition, because this logging is part of the application, a large amount of application context is available to the provenance system.

Third, we added provenance logging to common infrastructure, such as Spring Integration. Spring Integration [6] is a Java framework for creating distributed applications by composing services and abstracting the message-oriented mechanisms that connect the services. This approach allows services to be written independently from each other focusing only on their inputs, functionality, and outputs. To add provenance gathering to Spring, we created a Spring Interceptor that creates provenance log entries for each message it sees. We also made a small change to the Spring XML configuration to add this interceptor to every channel so that we could gather provenance information for every communication between services. More details on the Spring Interceptor can be found in the full technical report. Details of the technical report can be found at https://bitbucket.org/crestlab/Curator-public.git.

4 Related Work

Curator is not the first provenance management toolkit. SPADE [4] and PLUS [3] are two systems that provide similar capabilities. These management systems aim to capture, encode, store, and query provenance information from a wide-range of systems. PLUS aims to support integration into applications by providing a library for reporting provenance, a SQL schema for storing provenance, and a query interface to access collected provenance. PLUS also provides a modified Mule enterprise service bus that captures provenance and a provenance reporting API.

SPADE is a similar system that aims to collect, encode, store, and query provenance data in distributed systems. The core of SPADE is a pluggable kernel with interfaces for collection, storage, and querying of provenance data. SPADE also considers how to handle provenance at multiple abstraction layers, with collection plugins to capture both operating system information from audit logs and from applications using a named pipe. Similar to PLUS, SPADE provides an API for applications to report provenance. In both SPADE and PLUS, the transfer of provenance data is handled separately from other logging messages.

The primary difference between Curator and the tools described above is that Curator integrates with existing logging infrastructures for distributed systems, instead of building an entirely parallel infrastructure to support the management of the provenance data. This provides the additional benefit that system designers and developers can integrate with their existing logging infrastructures. This also reduces the complexity of Curator, since it is not responsible for log management. Our initial version of Curator acted as a centralized log management system for provenance data, making the core of Curator large and complex.

ProvToolbox [7] is a Java library for working with provenance in the W3C-PROV format. It defines Java classes to represent provenance documents. ProvToolbox also provides functionality to convert between Java provenance documents and the W3C PROV formats (PROV-XML, PROV-JSON, PROV-N). Finally, it can generate images of provenance graphs using Graphviz [8]. However, we found that ProvToolbox didn't fulfill all of our needs. First, ProvToolbox

[6] https://projects.spring.io/spring-integration
[7] http://lucmoreau.github.io/ProvToolbox
[8] http://www.graphviz.org

depends on a large number of packages making it challenging to integrate with microservices that have their own, sometimes conflicting dependencies. Second, we desired an in-memory graph representation of provenance information that we could easily search and traverse and that would match the representation that we store in databases, which ProvToolbox does not provide. Third, we wanted a simpler API based on the expectation that we will need to implement that API in a variety of programming languages. These reasons led us to develop the `Curator` representation for provenance graphs and support for serializing and deserializing those graphs to W3C-PROV formats.

The Provenance-Aware Storage System, PASS [12] supports an API for disclosing provenance called the Disclosed Provenance API, or DPAPI. The Core Provenance Library, CPL [10] also provides an API for reporting provenance from userspace applications. The PASS kernel and userspace applications utilize the DPAPI to integrate provenance from the OS and from the applications into a single provenance record for the system. Unlike ProvToolbox, the DPAPI and CPL do not require a particular specification. Instead, they are adaptable to different specifications. However, the DPAPI also expects the system to be running the PASS kernel, which may not be feasible for all applications.

5 Future Work

Several open challenges still exist that Curator does not currently address. First, security of collected provenanve data. Sevreral systems have looked at ways to provide secure provenance [1, 6] and it remains to be seen how Curator integrates with these. Second, linking high-level, semantically rich provenance to low-level system provenance continues to be a challenge for all provenance systems that do not provide explicit linking between high- and low-level events [12]. As we continue to leverage Curator, these enhanced capabilities will be explored further.

Our research and development roadmap also includes adding a number of analytics and visualization capabilities to `Curator`. We plan to add support for streaming analytics so that provenance data is analyzed as it arrives. In both of these areas, we will follow our existing approach and use existing systems to provide the primary functionality (e.g. Storm [9]) with `Curator` providing provenance-specific components to use with those systems. Future work towards visualization of provenance data includes an interactive provenance graph, allowing users to navigate through provenance data visually, as well as to filter and search the provenance data to find nodes or portions of the graph that match given criteria.

6 Conclusion

This paper described `Curator`, a toolkit designed to make it easier to integrate provenance into distributed systems. `Curator` provides tools for logging provenance, retrieving provenance from logs, storing provenance in databases, and analyzing and visualizing provenance. `Curator` is a composable set of tools where a developer can select the tools, such as provenance loggers, that best match their system. This paper also provided examples of how we used `Curator` in distributed systems. We have found that the design of `Curator` increases the likelihood that developers will add provenance to a system because it lowers the impact on that system compared to previous approaches. We also found that `Curator` reduces the effort it takes to add provenance to a system. Existing systems for generating and managing provenance information do not have these advantages.

[9] https://storm.apache.org

Availability

The source code for Curator can be found at https://bitbucket.org/crestlab/Curator-public.git.

Acknowledgments

The authors would like to thank the anonymous reviewers for their helpful feedback. They would also like to acknowlege the hard work of the Secure Resilient Systems and Technology Group at MIT Lincoln Laboratory, many of whom had discussions with the authors about what would make Curator most useful for their work.

References

[1] Adam Bates, Dave Tian, Kevin R.B. Butler, and Thomas Moyer. 2015. Trustworthy Whole-System Provenance for the Linux Kernel. In *24th USENIX Security Symposium (USENIX Security 15)*. USENIX Association, Washington, D.C. https://www.usenix.org/conference/usenixsecurity15/technical-sessions/presentation/bates

[2] Frank Capobianco, Christian Skalka, and Trent Jaeger. 2017. AccessProv: Tracking the Provenance of Access Control Decisions. In *9th International Workshop on Theory and Practice of Provenance*.

[3] A. Chapman, B.T. Blaustein, L. Seligman, and M.D. Allen. 2011. PLUS: A provenance manager for integrated information. In *Information Reuse and Integration (IRI), 2011 IEEE International Conference on*. 269–275. https://doi.org/10.1109/IRI.2011.6009558

[4] Ashish Gehani and Dawood Tariq. 2012. SPADE: Support for Provenance Auditing in Distributed Environments. In *Middleware 2012*, Priya Narasimhan and Peter Triantafillou (Eds.). Lecture Notes in Computer Science, Vol. 7662. Springer Berlin Heidelberg, 101–120. https://doi.org/10.1007/978-3-642-35170-9_6

[5] Paul Groth and Luc Moreau. 2013. PROV-Overview: An Overview of the PROV Family of Documents. https://www.w3.org/TR/prov-overview/. (April 2013).

[6] Ragib Hasan, Radu Sion, and Marianne Winslett. 2009. The Case of the Fake Picasso: Preventing History Forgery with Secure Provenance.. In *FAST* (2009-03-03). USENIX, 1–14. http://dblp.uni-trier.de/db/conf/fast/fast2009.html#HasanSW09

[7] Wajih Ul Hassan, Adam Bates, and Thomas Moyer. 2018. Towards Scalable Cluster Auditing through Grammatical Inference over Provenance Graphs. In *Network and Distributed System Security Symposium, NDSS 2018*.

[8] Trung Dong Huynh, Michael O. Jewell, Amir Sezavar Keshavarz, Danius T. Michaelides, Huanjia Yang, and Luc Moreau. 2013. The PROV-JSON Serialization: A JSON Represntation for the PROV Data Model. https://www.w3.org/Submission/2013/SUBM-prov-json-20130424/. (April 2013).

[9] Kyu Hyung Lee, Xiangyu Zhang, and Dongyan Xu. 2013. High Accuracy Attack Provenance via Binary-based Execution Partition.. In *NDSS*. The Internet Society. http://dblp.uni-trier.de/db/conf/ndss/ndss2013.html#LeeZX13

[10] Peter Macko and Margo Seltzer. 2012. A general-purpose provenance library. In *4th USENIX conference on Theory and Practice of Provenance (TaPP'12)*. USENIX Association, Berkeley, CA, USA, 6–6. http://dl.acm.org/citation.cfm?id=2342875.2342881

[11] Thomas Moyer and Vijay Gadepally. 2016. High-throughput ingest of data provenance records into Accumulo. In *High Performance Extreme Computing Conference (HPEC), 2016 IEEE*. IEEE, 1–6.

[12] Kiran-Kumar Muniswamy-Reddy, Uri Braun, David A. Holland, Peter Macko, Diana Maclean, Daniel Margo, Margo Seltzer, and Robin Smogor. 2009. Layering in Provenance Systems. In *2009 Conference on USENIX Annual Technical Conference (USENIX'09)*. USENIX Association, Berkeley, CA, USA, 10–10. http://dl.acm.org/citation.cfm?id=1855807.1855817

[13] Wenchao Zhou, Qiong Fei, Arjun Narayan, Andreas Haeberlen, Boon Thau Loo, and Micah Sherr. 2011. Secure Network Provenance. In *23rd ACM Symposium on Operating Systems Principles (SOSP)*.