Provenance Auditing in the GWDG

Maxim Barnstorf
Georg-August-Universität Göttingen
maxim.barnstorf@stud.uni-goettingen.de

October 19, 2025

1 Introduction

The foundation of the scientific method is validating or disproving hypotheses with data gathered from experiments. As our ability to gather, store, and process data steadily increases, the opportunities for using this data in that process increase as well. This in turn leads to experiments involving more, and more complex data and more sophisticated computations to make use of it. Take, for instance, the recent advancements in machine learning. The computations to train a big model are very resource intensive and involve large amounts of training data. The fact that we are now able to perform these calculations is completely revolutionizing the way research is being performed in many areas.

With these computations becoming more advanced, it becomes simutaniously much harder and more important to be able to reproduce the results of those computations. It is very easy to lose sight of the used data if it is split up into millions of individual files. And every file matters. It is clear that even a slight difference in the source-data, be it due to missing or different files or a even slightly different content, can cause major changes in the overall interpretation of the dataset. Reproducibility is most vital for scientific experiments, so this represents a real issue. The same goes for the software which is used to interpret the data. Often, multiple programs are used and their version matters. In extreme but not necessarily rare cases, the incorrect version will simply make it impossible to perform the interpretation because of compatibility issues.

This issue can be addressed by provenance auditing services. With the help of these tools, dependencies can be monitored so the user can later access information about the dependencies used and understand how the computation was carried out.

However, most existing provenance auditing services do not offer the right functionality to do this at scale and on High-Performance Computing (HPC) clusters, which are designed to efficiently carry out scientific computations, addressing the demand for more capable computing infrastructure resulting from the shift to more data-intensive experiments. The main contributions of this paper are:

- **Gap analysis of provenance auditing services:** An analysis of the characteristics a provenance auditing service ideally should have for HPC and how existing tools compare to that.
- **Provenance auditing service taxonomy:** Since the auditing can be performed in different ways, a taxonomy is proposed that aims to categorize each service depending on how it obtains the data.

2 The ideal provenance auditing tool

In order to understand which tools could be used, it helps to understand what the ideal tool for HPC would look like. Very broadly it comes down to the following 4 properties:

- **Quality of the provenance data:** It would track not just the input and output data but also which part of the input produced which part of the output, the software that has been used, including its versions, the overall environment like the version of the kernel, and the user who is carrying out the computation. It would also be able to collect this information about containerized computations, so Docker or Kubernetes would not be an obstacle. It ultimately comes down to how close the recorded information gets to 100% reproducibility.
- **Scope of applications:** It would work regardless of the programming languages, workflow managers, containers, etc. used for the computations and can therefore be applied everywhere.
- **Transparency:** It would be transparent to the user, meaning the user would not notice it and does not have to do anything for it to work. Otherwise errors would be introduced, it would take valuable time away from the scientists, and most importantly, it would decrease adoption because the benefits of provenance auditing are often only understood in hinsight.
- **Applicability:** The service would be broadly and systematically applicable across every computation. This is ultimately a function of the scope of applications and the transparency to the user.
- **Minimal overhead:** It would have minimal time and space overhead, meaning that the auditing barely slows down the computation. The biggest factor that slows down computations is processing overhead, but the file and folder read and write speed also factors in.

These properties are not binary. Rather, every tool fulfills each criterion to a certain extent. Unfortunately, provenance auditing is highly complex, and despite many attempts over the last 30 years, no tool exists that fits the above description. It is always about tradeoffs, for example, between granularity and time overhead or between specificity of the provenance data and the range of applications. More precise data will take more overhead to be generated. A tool supporting a broad range of applications will never offer specific information for a certain library.

3 Provenance auditing categories

There are many ways to collect provenance auditing, and that is reflected in the variety of tools that exist today. Many of them are specialized for a certain workflow. Others for certain languages. Some are completely transparent and independent of the computation. Others rely on the user to integrate the service into their code. To gain a better overview, here the different tools will be categorized:

Provenance APIs can provide a great deal of control. Here the user has to manually call the API from within the code to report back to the provenance tool, thus rewriting the application.

Kernel instrumentation enables the highest level of compatibility and transparency to be reached. Here, the kernel itself is monitored in order to inspect the users system calls. Every time a file is read and written to or a script is being executed, a call to the kernel has to be

made. This works regardless of the workflow, and the user never even has to know that it exists.

Library wrappers work by wrapping the original library and auditing the calls made to it. It can be better tailored to the particular functionality of the library and could give specific insights more flexible tools can not.

Compute platform based tools use workflows to access the provenance data. Most tools in this category are workflow based services. Many of the workflow-systems used have provenance features as part of their original design and are specifically made for provenance auditing. The compute platform term is used as an umbrella term here and also encompasses services hooking into compute frameworks, namely RAMP [1] and code frameworks, specifically ProvBook [2].

Other tools exist. Another option is to augment the language, injecting API calls to the provenance auditing system into the code by parsing a specific language, usually SQL. There are even modified language interpreters like CXXR [3] for R which then has provenance auditing integrated into the R interpreter, recording the data while interpreting the R code.

3.1 Categorization of existing tools

Here are a selection of existing provenance auditing tools and the categories they fit in:

- **Compute platform:** ZOOM [4], Vistralis [5], Kepler [6], PLUS [7], REDUX [8], Chimera [9], myGrid/Taverna [10], Dataprov [11], Wings-Pegasus [12], RAMP [1], ProvBook [2], Karma [13]
- **Augmented language:** Lipstick [14], Perm-GProM [15], Cui 2000 [16]
- **Provenance language:** Swift [17]
- **API based:** PReServ [18], ProvLake [19], AiiDA [20], Core Provenance Library [21], RDataTracker [22]
- **Language Interpreters:** CXXR [3]
- **User space events:** BURRITO [23]
- **Library Wrappers:** SPROV [24], PROV-IO [25]
- **Kernel instrumentation:** LPS [26], SPADEv2 [27], PASSv2 [28], ES3 [29], CamFlow [30]

4 Discussion of applicability

The tool must be compatible with existing software in the HPC environment. The user should not be forced to use a custom workflow manager or a programming language specifically for the provenance auditing. This would severely limit adoption. Additionally, transparency will be factored into applicability as well. If the users need to exert great effort to learn and integrate the respective services into their computations, it will heavily reduce adoption, important resources that should be spent on research will be misdirected, and it leaves room for inevitable errors. Therefore, before determining the performance of a given service, this segment analyzes the compatibility of each tool and its transparency, respectively. Good performance does not help if the service is never being used.

4.1 Compute platform

For workflow-specific provenance auditing tools, while the method is transparent, the main issue is always that the respective workflow managers are not commonly used. ZOOM [4], Vistralis [5], Kepler [6], myGrid/Taverna [10], Wings-Pegasus [12] and Chimera [9] are custom-made workflow managers that include provenance as their central feature. As a result, they are

not suitable. REDUX [8] uses the Windows Workflow Engine. Karma [13] was developed for LEAD used for scientific workflows related to weather simulations. Both are not used in the GWDG HPC environment.

There are services in this category that are built for platforms widely used. RAMP [1] uses Hadoop's MapReduce. Hadoop is more popular, and that could justify implementing it. Dataprov [11] is compatible with Snakemake which is excellent because Snakemake is quite commonly used. For this reason, Dataprov could actually be useful. ProvBook [2] is meant to be used with Jupyter Notebook, which is ubiquitous. As a result, ProvBook is also worth evaluating further.

4.2 API-based

All the API-based solutions require the user to heavily modify their existing scripts to include calls to the provenance API. This kind of overhead is just not scalable and only useful in niche applications. The API-based method is the least transparent of all. In very specific cases it can offer control that is otherwise impossible, but it is more suitable for individuals who decide to do this on their own terms rather than system wide.

Usually these APIs provide libraries for one or several languages. Python is the most popular language in HPC, so when judging the scope of applications, APIs made for Python will be adopted more frequently, which are AiiDA [20] and ProvLake [19]. The other API-based tools (PReServ [18], Core Provenance Library [21] and RDataTracker [22]) are meant to be used with other languages. Finally, to reiterate, the high user overhead ultimately makes none of them deployable at scale regardless of the language they are designed for.

4.3 Library wrappers

Library wrappers might sound like their range of applications is very restricted, but this is not necessarily the case. SPROV [24] for example, wraps functions inside the C I/O library stdio.h. This means it can be used for C and C++ programs. Questions about transparency can also be raised. After all, the original library needs to be replaced with the wrapper first. But this can be done by the system administrators. PROV-IO [25] is built for the system administrators to embed it into existing C/C++ libraries. That does require significant effort, especially across multiple libraries. But after that is done, it will not create any further overhead for users. PROV-IO also supports programs using HDF5 transparently.

4.4 Kernel instrumentation

Kernel instrumentation services are, by design, transparent and the most versatile because they are working at the core of the OS. Here the primary factor restricting usage is the kernel version it was designed for. For example, PASSv2 [28] was first published in 2006, and the Linux kernel has changed since. The newer version, PASSv2, is incompatible with the latest version as well. Ideally the system would be built with future maintenance in mind so it can be continually adjusted to changes in the Linux kernel. This was one of the main motivations behind CamFlow [30] which was made by one of the authors of PASSv2 and is currently the state-of-the-art.

Since the main advantage of the kernel instrumentation method is to be usable independently of the programming language or workflow and without the user having to interact with it, the kernel compatibility is the only real obstacle. ES3 [29] is from 2008 but since it is using strace, it

should still be compatible. SPADEv2 [27], LPS [26] and Camflow are all modern and compatible with modern Linux kernels.

4.5 Other

Similarly the technique of augmenting languages to store provenance is also only applied to specific languages. This method appears to never be used with conventional languages like Python, likely because of the involvement of libraries and general complexity. Perm-GProM [15] and Cui 2000 [16] primarily use SQL. They receive the queries and directly inject the provenance auditing. This is an obvious and low-overhead method, not just for the user but also for the processor. But SQL is not helpful for typical HPC computations. Lipstick [14] parses and modifies Pig Latin to add provenance auditing. Again, Pig Latin is not commonly used.

The Swift scripting language [17] would have to be learned and adopted just for the auditing, so it is not useful either. And additionally, it would severely limit what the user could do and would therefore be useless even if the users were willing to learn Swift.

CXXR [3] is a modified R interpreter that includes provenance auditing that way. It seems rather niche, and most importantly, a language-specific tool is not widely usable enough unless it is for Python.

BURRITO [23] has a variety of plugins for Bash and even the Firefox browser to collect information that way. Here, a primary focus is recording the methods the scientist is employing with the idea of providing insight into the research methodology of the user rather than regular provenance. This is also not what is required.

4.6 Summary

Therefore the different services can be categorized according to the scope of their applicability:

- **Niche/Not-applicable:** ZOOM [4], Vistralis [5], Kepler [6], PLUS [7], REDUX [8], Chimera [9], myGrid/Taverna [10], Wings-Pegasus [12], Lipstick [14], Perm-GProM [15], Cui 2000 [16], Swift [17], Karma [13], PReServ [18], ProvLake [19], AiiDA [20], Core Provenance Library [21], RDataTracker [22], CXXR [3], BURRITO [23], PASSv2 [28]
- **Applicable in common applications:** SPROV [24], PROV-IO [25], Dataprov [11], ProvBook [2], RAMP [1]
- **Universally applicable:** LPS [26], SPADEv2 [27], ES3 [29], CamFlow [30]

In summary, RAMP, Dataprov, ProvBook, SPROV, PROV-IO, SPADEv2, ES3, LPS and CamFlow could be worth implementing.

5 Discussion of Overhead

How much overhead is the service creating? What kind of overhead is being created? It is not just about the increased demand for processing power. The speed at which files and directories are opened, closed, and modified, as well as how much RAM is needed, can also be important. Finally, disk space requirements could also be of note but not as relevant as the other metrics. This section will be dealing with time overhead specifically because that is the most critical.

A time overhead comparison is really difficult to do. Different workloads are being used for the evaluation in the respective papers. Often, the overhead will only be presented with broad ranges depending on the operations audited, and it is not completely clear how the level of detail compares between tools. Therefore, rather than aiming for precision, the goal in this

comparison is to get a rough estimate. Since the differences between the tools can be very significant, even if one estimate over- or understates the true performance by, say, 50%, it is not critical. This section deals more with comparing the orders of magnitude rather than precise figures. This section focuses on the tools which are applicable in HPC. A full list is provided in the evaluation.

Dataprov	ProvBook	RAMP	SPROV	PROV-IO	ES3	SPADEv2	CamFlow	LPS
0%	0%	26-75%	12-16%	0.02-11%	-	12%	12-23%	1%

Table 1: Time overhead of selected services.

Dataprov [11] just does a quick Snakemake dry run to gather the data, which should take few resources, so it should not be a concern. The ProvBook [2] paper did not provide any information about its overhead. It should also be negligible, though, as it simply needs to record the input code and the output for it to function. Therefore, it is perfectly reasonable for DataProv and ProvBook not to provide any estimates. RAMP [1] has a very high overhead of 26-75%. Tasks using MapReduce can often take a significant amount of time when applied to big datasets, and that time overhead will be significant. It is therefore not suitable to be used for every MapReduce task systematically.

LPS [26] is specifically built for HPC, and the main reason behind its creation is the high overhead of comparable tools. Of note is that LPS includes an automatic system for throttling syscall probes when a certain threshold of I/O operations is reached. It is not possible to recognize how many I/O operations will be performed in advance, and even if the user would know, it would undermine transparency to ask them to manually adjust this granularity. LPS is the only service in this list that has this capability. Other kernel instrumentation tools do not prioritize performance enough to include it. It is important to emphasize that adjusting granularity at runtime is a unique feature of kernel instrumentation tools and that services using most other methods are unable to do this, although exceptions are possible. Due to this, no matter the computation, LPS can always stay below 1% overhead.

PROV-IO [25] also has acceptable performance, the 11% is only for HDF5, and the paper generally reports overhead of 3.5%. In one experiment the authors ran where it was integrated into a C library, the overhead did not exceed 0.02%.

Meanwhile, SPROV [24], SPADEv2 [27], and CamFlow [30] are all in the 10-20% range. The CamFlow authors only provided direct performance estimates for specific kernel operations, not relative to the program executed. However they did state that it is in line with SPADEv2 and PASSv2. The authors of ES3 [29] also never provided a time overhead estimate, but considering the overhead of the other kernel instrumentation tools and that good performance was never a criterion for ES3, it is likely similar as well. None of these tools were explicitly built for HPC and it is reflected in this large time overhead. That kind of overhead is not acceptable on an HPC system. It is very hard to justify a 10% minimum overhead given how expensive data center hardware and electricity are, no matter how beneficial the provenance auditing might be.

The overhead of the workflow manager-based systems that have provenance auditing included, as well as the Swift provenance language, cannot be measured, as it is part of the core of the respective system and not separable. As for APIs, most of them were unfortunately not evaluated for performance. Lastly, serveral papers provide overhead assessments in differet formats, one example provided earlier is CamFlow. While this can make sense in some circumstances, these measurements are left out in the final comparison. To summarize, when considering the overhead of sufficiently applicable systems, only Dataprov, ProvBook, LPS, and

PROV-IO are feasible.

6 Evaluation

Considering the criteria together, 4 of the 30 tools, Dataprov, ProvBook, LPS and PROV-IO, have a reasonably low overhead and can at least be somewhat broadly used considering both transparency and the application scope:

System	Overhead	Transparent	Scope	Applicability	Category
AiiDA	-	No	◐	○	API
BURRITO	-	Yes	○	○	User Space
CamFlow	12–23%	Yes	●	●	Kernel Instrumentation
Chimera	/	Yes	○	○	Compute Platform
CPL	-	No	○	○	API
Cui 2000	-	Yes	○	○	Augmented Language
CXXR	-	Yes	○	○	Language Interpreter
Dataprov	None	Yes	◐	◐	Compute Platform
ES3	-	Yes	●	●	Kernel Instrumentation
Kepler	/	Yes	○	○	Compute Platform
Karma	-	Yes	○	○	Compute Platform
Lipstick	16-35%	Yes	○	○	Augmented Language
LPS	1%	Yes	●	●	Kernel Instrumentation
myGrid/Taverna	-	Yes	○	○	Compute Platform
PASSv2	23%	Yes	○	○	Kernel Instrumentation
Perm-GProM	-	Yes	○	○	Augmented Language
PLUS	/	Yes	○	○	Compute Platform
PReServ	10%	No	○	○	API
PROV-IO	0.02–11%	Yes	◐	◐	Library Wrapper
ProvBook	None	Yes	◐	◐	Compute Platform
ProvLake	1%	No	◐	○	API
RAMP	26–75%	Yes	◐	◐	Compute Platform
RDataTracker	-	No	○	○	API
REDUX	/	Yes	○	○	Compute Platform
SPADEv2	12%	Yes	●	●	Kernel Instrumentation
SPROV	12–16%	Yes	◐	◐	Library Wrapper
Swift	/	Yes	○	○	Provenance Language
VisTrails	/	Yes	○	○	Compute Platform
Wings-Pegasus	/	Yes	○	○	Compute Platform
ZOOM	/	Yes	○	○	Compute Platform

Table 2: Performance overhead, transparency, scope, applicability, and category of all provenance systems.

Legend: Overhead: - = Not measured, / = Not measurable
Scope, Applicability: ● = Broad, ◐ = Partial, ○ = Limited.

6.1 LPS

LPS [26] uses Systemtap [31] to probe the kernel for three categories which will establish a complete history of the files the process read and modified:

- **Execution:** LPS probes the kernel for syscalls related to process creation, execution and termination.
- **File access:** open, close, read and write calls are being recorded to keep track of file accesses and changes.

- **Metadata:** rename, link, unlink and delete operations are being kept track of.

LPS identifies which calls to save by using the original process as a root node and building a tree from there with child processes as nodes. In HPC, users use ssh and shell to launch their computations from login nodes. In compute nodes, computations are executed either through shell or other runtime libraries. LPS will recognize these origin processes, build this execution tree, and keep track of file changes invoked by any child process. This way, even parallelization will not be an obstacle, as everything can be tracked back to the original root process.

The syscalls do not reveal the actual changes made to the files. Therefore, it does not help to debug or understand a script. The objective is reproducibility by keeping track of the inputs, and the information provided by these operations is sufficient to accomplish that.

LPS demonstrates that kernel instrumentation tools can be very efficient. While it has never been published, it still demonstrates that it will be more cost-effective to build a similar system compared to incurring the large overhead created by other, less optimized kernel instrumentation systems listed in this category, considering how high the cost of data center hardware and the electricity consumed is.

6.2 PROV-IO

PROV-IO [25] can actually be used fully transparently. It can be embedded into any C/C++ application, which has some very unique benefits. Depending on the library, very different information is relevant and needs to be tracked. Kernel instrumentation tools could never track this information, and neither could workflow manager-based services beyond data on the workflow itself. The only other method that allows for such tailored data is provenance APIs. But this method is the only one allowing it transparently. It follows that this tool could uniquely fill an important niche. Unlike LPS it has also been published, although this has been done for development purposes, not large scale application in production.

Overall the burden here lies on the system administrators that need to understand which provenance data is important for which library and who have to actually modify the existing library. This is a constant effort, with libraries receiving frequent updates and with different libraries becoming more or less popular. This is really the main drawback for PROV-IO and quite a severe one. That unfortunately also makes it unsuitable for the GWDG HPC environment.

6.3 ProvBook

ProvBook [2] simply saves all the code executed for each block and its output and allows the user to view previous inputs and outputs conveniently within Jupyter Notebook. It makes it possible to scroll through the entire history with a slider. This is practical and immediately useful. Minimizing the friction the user has with viewing the provenance data is critical for adoption. The notebooks with the data can also be exported to RDF. This file can later be converted back into a notebook as well. Installation is very convenient as a Jupyter Notebook extension and should not be an issue for system administrators. Overall it is quite useful within the scope of Jupyter Notebook and could be worth implementing given how many people use it. It is also available on Github.

6.4 Dataprov

Finally, Dataprov [11] provides a reasonable amount of provenance auditing without any relevant time overhead and complete transparency for Snakemake. Dataprov will record all input and

output files without attributing them to the respective step in which they were used or produced. It will also record programs used and their versions, as well as the kernel version and information about the user, which they are required to enter. A lot of what the tool achieves can also be achieved by simply backing up the Snakemake file at each run. But with Dataprov it is formatted in XML which could then be used to store it in a database in the future. A visualization for said data would also have to be programmed. It is significantly less complex than most tools listed, but that is part of what makes it useful. Fortunately, Dataprov has also been published and it works, albeit only with specific dependencies. It is the only service that can be applied to a reasonable percentage of computations, has a low overhead, and is transparent.

7 Conclusion and Outlook

To conclude, when looking at transparency, scope of applications, overhead to the user, and overhead to the system administrators, LPS [26], Dataprov [11] and ProvBook [2] could be worth implementing. Specifically, LPS could have great potential, as it can be used system-wide and can keep track of all input files. While it is not publicly available, there is much to learn from the concepts presented in the paper, specifically in terms of efficiency and auditing the provenance holistically across the child processes invoked by the original process.

As future work, Dataprov and ProvBook could be implemented in a real HPC data center. This will also involve storing the collected data in a database that can be queried and respond with human-readable output. The input and output data referenced by the provenance auditing data could also automatically be indexed and stored using the GWDG GöDL Data Catalog service [32] to enable searching and organizational capabilities, as this data can be notoriously hard to keep track of and organize. Additionally, a system with a similar approach to LPS could be developed and integrated as well.

References

[1] H. Park, R. Ikeda, and J. Widom, "Ramp: a system for capturing and tracing provenance in mapreduce workflows," Proc. VLDB Endow., vol. 4, p. 1351–1354, Aug. 2011.

[2] S. Samuel and B. König-Ries, "Provbook: Provenance-based semantic enrichment of interactive notebooks for reproducibility," in Proceedings of the 17th International Semantic Web Conference (ISWC 2018) - Demo and Poster Track, vol. 2180 of CEUR Workshop Proceedings, (Monterey, California, USA), pp. 57–60, 2018.

[3] A. Runnalls and C. Silles, "Provenance tracking in r," in Provenance and Annotation of Data and Processes (P. Groth and J. Frew, eds.), (Berlin, Heidelberg), pp. 237–239, Springer Berlin Heidelberg, 2012.

[4] S. Cohen-Boulakia, O. Biton, S. Cohen, and S. Davidson, "Addressing the provenance challenge using zoom," Concurr. Comput. : Pract. Exper., vol. 20, p. 497–506, Apr. 2008.

[5] L. Bavoil, S. Callahan, P. Crossno, J. Freire, C. Scheidegger, C. Silva, and H. Vo, "Vistrails: enabling interactive multiple-view visualizations," in VIS 05. IEEE Visualization, 2005., pp. 135–142, 2005.

[6] S. Bowers, T. M. McPhillips, and B. Ludäscher, "Provenance in collection-oriented scientific workflows," Concurrency and Computation: Practice and Experience, vol. 20, no. 5, pp. 519–529, 2008.

[7] A. Chapman, M. Allen, B. Blaustein, L. Seligman, P. Profiler, M. Morse, and A. Rosenthal, "Plus: Provenance for life, the universe and stuff," Proceedings of the VLDB Endowment, 01 2010.

[8] R. S. Barga and L. A. Digiampietri, "Automatic capture and efficient storage of e-science experiment provenance," Concurr. Comput. : Pract. Exper., vol. 20, p. 419–429, Apr. 2008.

[9] I. T. Foster, J.-S. Vöckler, M. Wilde, and Y. Zhao, "Chimera: Avirtual data system for representing, querying, and automating data derivation," in Proceedings of the 14th International Conference on Scientific and Statistical Database Management, SSDBM '02, (USA), p. 37–46, IEEE Computer Society, 2002.

[10] T. Oinn, M. Addis, J. Ferris, D. Marvin, M. Senger, M. Greenwood, T. Carver, K. Glover, M. R. Pocock, A. Wipat, and P. Li, "Taverna: a tool for the composition and enactment of bioinformatics workflows," Bioinformatics, vol. 20, p. 3045–3054, Nov. 2004.

[11] F. Bartusch, M. Hanussek, and J. Krüger, "Automatic generation of provenance metadata during execution of scientific workflows," in IWSG, 2018.

[12] E. Deelman, G. Singh, M.-H. Su, J. Blythe, Y. Gil, C. Kesselman, G. Mehta, K. Vahi, G. B. Berriman, J. Good, A. Laity, J. C. Jacob, and D. S. Katz, "Pegasus: A framework for mapping complex scientific workflows onto distributed systems," Scientific Programming, vol. 13, no. 3, p. 128026, 2005.

[13] Y. L. Simmhan, B. Plale, and D. Gannon, "A framework for collecting provenance in data-centric scientific workflows," in 2006 IEEE International Conference on Web Services (ICWS'06), pp. 427–436, 2006.

[14] Y. Amsterdamer, S. B. Davidson, D. Deutch, T. Milo, J. Stoyanovich, and V. Tannen, "Putting lipstick on pig: enabling database-style workflow provenance," Proc. VLDB Endow., vol. 5, p. 346–357, Dec. 2011.

[15] B. S. Arab, D. Gawlick, V. Radhakrishnan, H. Guo, and B. Glavic, "A generic provenance middleware for database queries, updates, and transactions," 06 2014.

[16] Y. Cui, J. Widom, and J. L. Wiener, "Tracing the lineage of view data in a warehousing environment," ACM Trans. Database Syst., vol. 25, p. 179–227, June 2000.

[17] Y. Zhao, M. Hategan, B. Clifford, I. Foster, G. von Laszewski, V. Nefedova, I. Raicu, T. Stef-Praun, and M. Wilde, "Swift: Fast, reliable, loosely coupled parallel computation," in 2007 IEEE Congress on Services (Services 2007), pp. 199–206, 2007.

[18] P. Groth, S. Miles, and L. Moreau, "Preserv: Provenance recording for services," 12 2010.

[19] L. G. Azevedo, R. Souza, R. M. Thiago, E. F. S. Soares, and M. F. Moreno, "Experiencing provlake to manage the data lineage of ai workflows," Anais Estendidos do XVI Simpósio Brasileiro de Sistemas de Informação (Anais Estendidos do SBSI 2020), 2020.

[20] S. Huber, S. Zoupanos, M. Uhrin, L. Talirz, L. Kahle, R. Häuselmann, D. Gresch, T. Müller, A. Yakutovich, C. Andersen, F. Ramirez, C. S. Adorf, F. Gargiulo, S. Kumbhar, E. Passaro, C. Johnston, A. Merkys, A. Cepellotti, N. Mounet, and G. Pizzi, "Aiida 1.0, a scalable computational infrastructure for automated reproducible workflows and data provenance," Scientific data, vol. 7, p. 300, 09 2020.

[21] P. Macko and M. Seltzer, "A general-purpose provenance library," in Proceedings of the 4th USENIX Conference on Theory and Practice of Provenance, TaPP'12, (USA), p. 6, USENIX Association, 2012.

[22] B. S. Lerner and E. R. Boose, "Rdatatracker and ddg explorer," in Revised Selected Papers of the 5th International Provenance and Annotation Workshop on Provenance and Annotation of Data and Processes - Volume 8628, IPAW 2014, (Berlin, Heidelberg), p. 288–290, Springer-Verlag, 2014.

[23] P. J. Guo and M. Seltzer, "Burrito: wrapping your lab notebook in computational infrastructure," in Proceedings of the 4th USENIX Conference on Theory and Practice of Provenance, TaPP'12, (USA), p. 7, USENIX Association, 2012.

[24] R. Hasan, R. Sion, and M. Winslett, "Preventing history forgery with secure provenance," ACM Trans. Storage, vol. 5, Dec. 2009.

[25] R. Han, S. Byna, H. Tang, B. Dong, and M. Zheng, "Prov-io: An i/o-centric provenance framework for scientific data on hpc systems," in Proceedings of the 31st International Symposium on High-Performance Parallel and Distributed Computing, HPDC '22, (New York, NY, USA), p. 213–226, Association for Computing Machinery, 2022.

[26] D. Dai, Y. Chen, P. Carns, J. Jenkins, and R. Ross, "Lightweight provenance service for high-performance computing," in 2017 26th International Conference on Parallel Architectures and Compilation Techniques (PACT), pp. 117–129, 2017.

[27] A. Gehani and D. Tariq, "Spade: support for provenance auditing in distributed environments," in Proceedings of the 13th International Middleware Conference, Middleware '12, (Berlin, Heidelberg), p. 101–120, Springer-Verlag, 2012.

[28] D. Holland, M. Seltzer, U. Braun, and K.-K. Muniswamy-Reddy, "Passing the provenance challenge," Concurrency and Computation: Practice and Experience, vol. 20, pp. 531–540, 04 2008.

[29] J. Frew and P. Slaughter, ES3: A Demonstration of Transparent Provenance for Scientific Computation, p. 200–207. Berlin, Heidelberg: Springer-Verlag, 2008.

[30] T. Pasquier, X. Han, M. Goldstein, T. Moyer, D. Eyers, M. Seltzer, and J. Bacon, "Practical whole-system provenance capture," in Proceedings of the 2017 Symposium on Cloud Computing, SoCC '17, (New York, NY, USA), p. 405–418, Association for Computing Machinery, 2017.

[31] SystemTap Developers, "Conditional probes." https://www.sourceware.org/systemtap/, 2025.

[32] GWDG, "GWDG GöDL: Data Indexing for HPC." https://docs.hpc.gwdg.de/services/g%C3%B6dl/index.html, 2025.