Engineering Choices for Open World Provenance

M. David Allen(✉), Adriane Chapman, and Barbara Blaustein

The MITRE Corporation, Mclean, USA
{dmallen, achapman, bblaustein}@mitre.org

Abstract. This work outlines engineering decisions required to support a provenance system in an open world where systems are not under any common control and use many different technologies. Real U.S. government applications have shown us the need for specialized identity techniques, flexible storage, scalability testing, protection of sensitive information, and customizable provenance queries. We analyze tradeoffs for approaches to each area, focusing more on maintaining graph connectivity and breadth of capture, rather than on fine-grained/detailed capture as in other works. We implement each technique in the PLUS system, test its real-time efficiency, and describe the results.

Keywords: Provenance · Lineage · Pedigree · System engineering

1 Introduction

All provenance systems to this point have been applied to "closed world" systems. As described in [12], a closed world system contains at least one of the following properties: The underlying application or systems are known in advance and provenance enabled; a provenance administrator has administrative privileges for the systems and applications in use; or full knowledge of either the data or processes is known in advance. These assumptions work very well for scientific applications [5, 15, 19, 27, 30], within relational databases [9, 14], and for specific applications [15]. However, the world of large-scale enterprises, as typified by our U.S. government sponsors, is much messier.

Our users typically operate in environments that involve computations distributed across personnel and systems in very large enterprises. Their interests usually do not lie with replication of results or very fine-grained provenance, but with more general queries whose purpose is to help users build trust that a particular dataset is appropriate for their use. Government sponsors are trying to exploit available assets from other government groups, so most users who wish to use novel datasets will eventually need to investigate the provenance of that information to determine its suitability for the mission at hand. In Sects. 2–6, we describe system design research that is required for functioning open-world provenance systems. Section 7 evaluates each proposed technique. We discuss related work and conclude in Sects. 8 and 9 respectively.

B. Ludäscher and B. Plale (Eds.): IPAW 2014, LNCS 8628, pp. 242–253, 2015.
DOI: 10.1007/978-3-319-16462-5_25

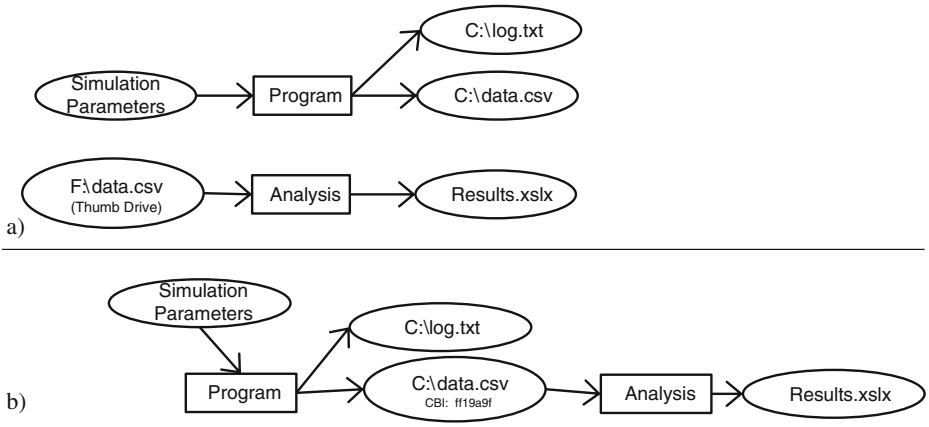

Fig. 1. (a) Two independent but related provenance graphs. (b) The correct provenance graph with the same data artifact from both graphs correctly identified.

2 Identity

The PROV Data Model [20, 28] uses URLs/URIs as identifiers for most things. If two entities have the same identifier then they are considered equal. We concur with this definition, but we have a frequent need to establish a common identifier that can be computed no matter what system/user/environment/organization is involved in the manipulation of the artifact. Figure 1 motivates the need for extensive identity capabilities in any open-world provenance system. A set of simulation parameters are given to a program that produces CSV data as an output (with a log). Days later, that CSV file is sent via a thumb drive or other unobservable method to a different user in another organization, across several network boundaries. That file is then saved to the user's hard disk and run through a separate analysis process. Figure 1(a) shows what can happen without an identification technique to establish common identity of data or process nodes in a provenance graph. An identification technique is fundamental, because we must concede that observing every provenance link without exception is unlikely. Our strategy, then, is to determine ways of identifying or "tagging" data in a way that will persist wherever and however the data might travel outside of our ability to observe it, resulting in a graph like that shown in Fig. 1(b).

Imposing an artifact naming or URI convention on provenance capture will not work in wide collaborations, and essentially no assumptions can be placed on the storage or transmission method of the data. Over a distributed system, it is entirely possible to see several, unrelated "Hello World Output.txt" files. Name, file size, owner, or other related metadata is useful to capture, but does not provide a sound basis for identification. Content does provide a basis: across many systems, the same file can be copied and the name changed, but the underlying information is the same: Alice's MyNotes.docx is the same as Bob's AliceNotes.docx in content.

When capturing and reporting provenance in complex environments, there may be **multiple, independent observations of the same thing** (sending system observes

transmission of *M;* receiving system Y observes the receipt of *M*). Additionally, data or processes may be observed via **different technical channels** (program generates a file *M* on disk; months later, user receives email with attachment *M*). These requirements create situations where disconnection and duplication occurs [4].

The solution is to adopt content- or context-bound identifiers. A content-bound identifier (CBI) is any identifier that is effectively computed as a function of the content of a data item. Content-bound identifiers permit two independent observers to identify the item the same way, even if they are ignorant of the existence of the other observers. Provenance reporting systems can use content-bound identifiers as a way of synchronizing multiple observers/reporting clients, and de-duplicating what would otherwise become redundant and disconnected. Context-bound identifiers are more suitable for tracking different program executions across different environments; at the moment, content-bound identification of data is most useful because data *is what is moving across machine boundaries*. Some computing environments such as Hadoop *move the processing to the data* because of data volumes. Invocations can be de-duplicated with context-bound identifiers; unfortunately unlike *content*-bound identifiers, what constitutes a good context-bound identifier will be different depending on the underlying computational system.

In the example above, we must establish that the file named data.csv and the thumbdrive data are the same document, as shown in Fig. 1(b). Using content-bound identifiers, all parties that touch the file will compute the same CBI, thus providing the proof that the provenance graphs are indeed joined.

There are many available options for cryptographic hash functions; most have some suitability for content-bound identifiers. This section is not an exhaustive review, but just a brief look at two very common functions, along with their pros and cons. MD5, first published in 1992, produces 128-bit digests (hashes), while SHA256 is an instance of the SHA-2 cryptographic hash function that produces 256-bit digests.

Hash functions for provenance identity should be evaluated in terms of three aspects: performance (data volume hashed in a given period of time), resistance to collision (likelihood that two different data items would have the same digest), and size (how much data the digest contains). In terms of these tradeoffs, SHA-256 is larger, more robust/resistant to collisions, and slower than MD5 (see Sect. 7.1). MD5 is discouraged for cryptographic applications [29] yet is still in wide use in environments where collisions are not a primary concern.

3 Storage

There are several options for storing provenance information. These include: relational [5, 6, 31]; flat file [23]; bound to the data itself [17]; graph-based [13, 18, 25]. These storage options are not mutually exclusive. It is possible to take information from a database, output provenance for a particular file and bind it to the data. However, the choice of which storage strategy to use for a provenance management system depends on factors including: technology required by provenance-using applications; directives and mandates; provenance information required (the required usage of the provenance information will dictate the style in which it is stored); network architecture (transmission between

different enclaves can be problematic, or even impossible); trust architecture (with many different government partners, trust issues may dictate that provenance needs to be hosted in a particular place, or not combined with other sources). At various times in our research system, PLUS, we have used relational databases, flat files, XML, and graph databases to store provenance.

XML: XML and other hierarchical document formats such as JSON and BSON are workable solutions, but an imperfect fit; the data model behind XML and JSON is fundamentally a tree, although XML languages that support directed graphs (i.e. GraphML) can help. XML is well-suited to expressing a subset of provenance graphs and data structures, but to express the full range of directed graphs, implementers will either fall back on the use of "pointers" (e.g. XML ID/IDREF) or data duplication within the document to express directed graphs without tree assumptions. In other words, the underlying model gets messy. In our experience, XML is useful as an interchange format, but not as a storage format because it complicates query.

Relational: For several years, our software used MySQL and PostgreSQL as a storage layer, providing us with extensive experience on the pros and cons of relational storage for provenance. Relational databases are attractive because of their wide adoption and mature tooling. We found, though, that the RDMBS made path-associative query extremely difficult. Storing provenance in an RDBMS typically involves a table of nodes and a table of edges. These designs are excellent for bulk query that does not require much edge traversal ("Fetch all provenance owned by Bob"), but tend to be very poor at path-associative queries ("Fetch all provenance that is between 2 and 5 steps downstream of *X*"). Path-associative queries typically end up being translated as dynamically constructed, variably recursive SQL queries that join nodes to edges. RDMBS rapidly pushes developers down the path of re-implementing basic graph techniques the RDBMS does not provide (e.g. shortest path algorithms) rather than exploiting known good implementations.

Graph DBs: Our findings over time have indicated that general purpose graph databases (such as Neo4J or, in principle, RDF triple stores) are by far the best fit for provenance, for two simple compelling reasons: (1) the graph model under the hood of a graph database is fundamentally a match for the core of provenance (a directed graph), and (2) graph databases will typically provide graph-oriented query languages (such as Cypher within Neo4J, or perhaps SPARQL within RDF triple stores) which greatly facilitate provenance queries. The negative aspect of graph databases is that because they are "naturally indexed" by relationships/edges, they do not perform as well on bulk queries mentioned above. While such bulk queries do have important uses, the most interesting and powerful provenance queries (see Sect. 8) typically are path-associative. This style of query emphasizes the strengths of graph query languages; an emphasis which plays to many of the weaknesses of other languages.

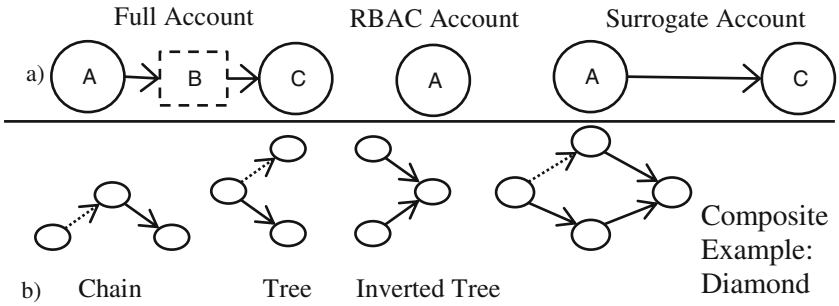

Fig. 2. (a) Example of provenance graph protection using surrogates. (b) Sample motif graphs.

4 Protection

Our US government sponsors are particularly concerned about protection of provenance information. Many times, materials and methods are more protected and sensitive than the resulting information. At a minimum, we must apply classic access control techniques to the provenance information [26]. However, classic access controls break provenance graphs. If a single node or edge within the provenance graph cannot be shown, then the provenance graph may be severely truncated. Consider the chain graph shown in Fig. 2(a). If the process B is sensitive and restricted, then the provenance graph showing descendants of A will only consist of A, instead of the richer graph. To this end, we have created surrogates as described in [10], in which we guarantee that protected nodes and edges will not be shown, but the utility of the graph is maximized by inserted surrogate nodes and edges.

The model we have adopted calls for permitting the attachment of various "privilege classes" to individual provenance nodes; users who attempt to access node information must demonstrate that they belong to the correct privilege class. Our notion of a privilege class is meant to subsume what we might otherwise refer to as a "role" or an "attribute" and, as such, the model is suitably general so that it can describe RBAC or ABAC. If users possess the right privilege classes, access is provided as normal. But if they do not, the surrogate algorithm seeks to provide access to as much information as possible, subject to user-configurable policies.

5 Testing

There have been previous efforts at creating provenance flows for testing. Of particular interest is the ProvBench effort [7] and the Provenance Challenge [1, 21]. ProvBench aims to distribute annotated provenance flows so that both the provenance and the intent of the overall workflow are understood within the dataset. We wish to exercise the system to ensure that it can work over any size or shape provenance graph in order to ensure all algorithms can function over bushy or sparse graphs. We do not claim that the generators discussed create provenance similar to the real observations; we target generators sufficiently tunable that they can mimic any form.

Motif Generators: We began with the observation that any provenance graph of any size and shape can be described as a conjunction of a set of smaller graph "primitives" or "motifs". Figure 2(b) shows the set of possible motifs that are generated by a motif generator. We built a motif generator that permits users to generate any number of randomly chosen motifs, with tunable connection parameters. In the most simple case, a motif generator might choose 100 random motifs; it would then choose a random node from within each motif and link it to a randomly chosen node from the next motif. Note that complex motifs such as a "diamond shape" can be created by joining simpler motifs (a tree, with an inverted tree).

Graph Simulation: Our graph simulator, DAGAholic, focuses on guaranteeing certain properties of the generated graph, including its size and edges, but does not generate any particular shape. DAGAholic is given parameters including number of nodes, proportions of data vs. invocations in the graph, and so on. Users specify a graph connectivity, which is the probability that a given node will be connected to something downstream in the provenance graph; 0.25 indicates that 25 % of nodes in the resulting graph will have an outbound provenance relationship. DAGAholic also has a rich set of options for protecting graphs. Because the determination to create a specific edge is based on a random number, the graphs, while generally conforming to the sparse/bushy objective, will be individually distinct.

Table 1. Examples of "canned queries released with PLUS"

Query	Description
Trace Taint Sources	Find all upstream "nodes marked as "tainted" or "corrupted" to determine quality of present information
Chain of Custody	Whose hands has this data passed through?
Time Span	The oldest item, the newest, and the time span between them
Distinct Sources	Number of distinct upstream sources: e.g., is this analysis based on five independent reports, or just one?

6 Output

We provide basic access to all provenance graphs via the Cypher query language, provided by Neo4J, to permit arbitrary query against stored provenance. Most users do not want to interact with the provenance graph or write queries. Instead, they have a goal such as checking the fitness for use of a particular item. One of our most common access patterns, then, is to establish the user's "fitness parameters" for information; for example, "I'm only interested in data less than 30 days old, which was processed by the Air Force, and went through System X". These parameters are then encoded as a set of pre-canned, but keyword-customizable, provenance queries [11]. Canned queries can then be arranged into dashboards for users which answer their questions, but do not require technical knowledge of provenance or query. Table 1 shows examples of canned queries currently packaged within PLUS, which can be combined into custom fitness assessments for new users.

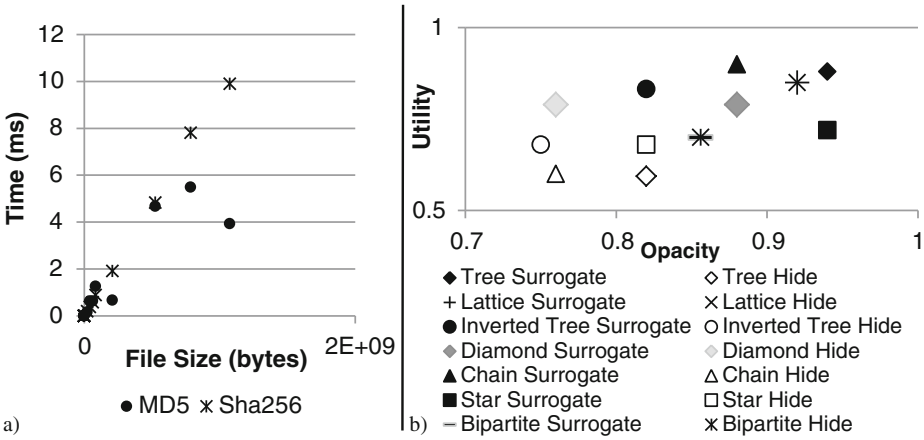

Fig. 3. (a) The time required to use hashing to generate a file identity using either MD5 or Sha256. (b) The tradeoff in utility for hiding or surrogating several sample graphs.

7 Implementation and Evaluation

Each problem discussed has been addressed within the PLUS system.[1] In this section, we use PLUS to demonstrate tradeoffs for these techniques and to illuminate the final system design decisions within PLUS.

7.1 Identity

Hashing is useful to define identity for artifacts from many different systems. In order to show that system performance is not unduly affected by incorporating hashing as an identifier, we created several data artifacts, of varying sizes, and ran them through each hashing function. Figure 3(a) shows the results; it only costs a few milliseconds on even the largest files. We also observe through these runs that the average memory size is 0 kB and the maximal memory size is between 2768 kB and 2928 kB, likely at process startup. Thus, we believe that using the hash value as an identifier is an acceptable method for identity, because its computational cost is low, and its memory requirements are small and fixed, irrespective of the input size. Because of concerns discussed earlier about compromises to MD5, we generally recommend the more secure and adopted SHA-2 algorithm, computing 256 bit hashes.

7.2 Storage

Using commercially available systems, such as MySQL and Neo4 J, the speed of each system is acceptable for a wide range of queries, but there are substantial performance

[1] https://github.com/plus-provenance/plus.

differences between different types of queries. Instead of comparing performance directly, we look at how easy or hard it is to perform operations specific to provenance. In order to provide an estimate on ease, we measure the lines of code required to create the functionality within the system, with either a relational or graph database backend. "Source lines of code" is unavoidably a coarse measure; in some cases minor differences may be accounted for by issues such as indentation style, volume of comments, and so on. These numbers are presented as our concrete implementation experience, and to provide a rough sense of the difficulty of implementation. We would expect alternative implementations to encounter the same set of issues we present in the discussion.

Our measures exclude the number of lines of code necessary for translating graph nodes into provenance objects (taking properties from the DB and putting them into java objects). In many applications, this serialization/deserialization of objects between the database and the object model is largely housekeeping work. Table 2 shows the lines of source code required for each storage implementation within PLUS.

Table 2. A snapshot of the code base required to support provenance manipulation.

Function	Description	Neo4j	MySQL
Load Graph from DB	Building database queries, iterating through results, returning a provenance collection consisting of nodes, edges, non-provenance edges, and actors	141	538
Trace Chain of Custody	Tracing through an entire provenance graph from some starting point, and extracting an ordered list of all owners of all data in the graph	85	638
Get Indirect Sources of Taint	Examine a particular provenance node and determine, at any distance upstream from the current node, if there is a marking indicating that one of its ancestors is "tainted" (e.g. has a problem, or is based on bad information as asserted by a user)	40	606
Arbitrary Graph Query	The ability for the user to formulate an arbitrary read-only query to traverse provenance graphs, returning any computable subset of provenance information	50	N/A

When loading a graph from a database, the resulting provenance collections form the basis of data presented to the user visually, and sent to other systems as reports. As a necessary prerequisite for so many other operations, loading a graph is probably the most common operation that a provenance system will do. In Neo4j, we use a traversal framework; the traverser does all of the work, and as the nodes and edges are returned, they are turned into provenance objects and added to a result collection. In MySQL, the traverser is custom-implemented code, essentially an iterator which fetches and joins nodes and edges. These issues apply to implementations that trace chains of custody and get indirect sources of taint.

Because data must be fetched via SQL when using MySQL, it is extremely difficult to implement arbitrary graph queries. With natively supported graph query languages

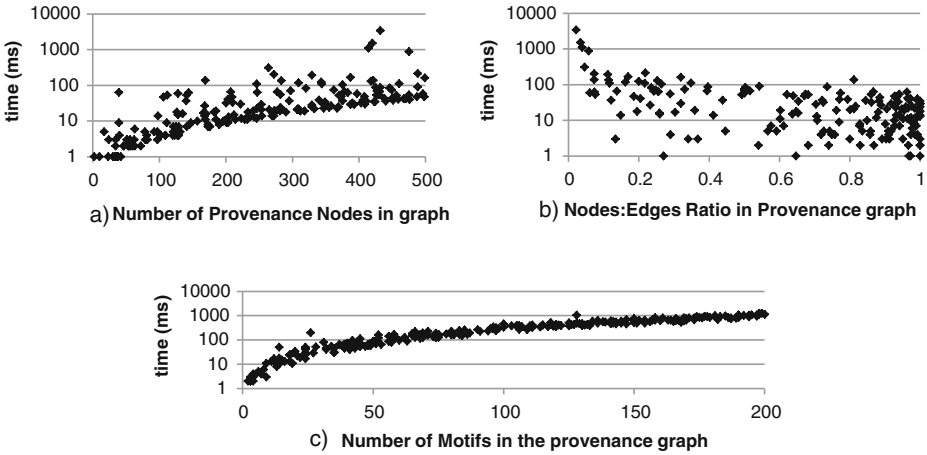

Fig. 4. (a) the time required to produce provenance test graphs, based on the number of nodes in the graph. (b) The time to create synthetic provenance graphs based on the node to edge ratio. (c) The time to create motif graphs based on the number of motifs in the final graph.

(such as Neo4j's cypher) most of the code is given over to simple housekeeping, such as query sanitization to apply safeguards to prevent the user from modifying or deleting data with a query. When using such graph databases, no new code is introduced; the user is simply given an interface to perform queries as they might with SQL, and a few utilities to visualize or report the results.

7.3 Protection

As discussed above, we need to protect sensitive data in provenance graphs, while maximizing the graphs' utility. We start with some basic graphs in classic patterns shown in Fig. 2(b), and "hide" the dashed edge within the graph. Figure 3(b) shows the difference between breaking the graph at the sensitive edge, as would occur with basic access control strategies, and surrogating. Since more of the graph is available for consumption with surrogating, the utility of the final provenance graph is higher.

7.4 Testing

The testing package included with the PLUS system has the ability to create generic provenance graphs of varying shapes and sizes. We are mostly concerned that the creation of a suite of provenance test graphs does not take undue time or resources. Figure 4 shows the time it takes to generate a set of provenance test graphs. Our test runs indicate that the cost in time is not exorbitant, allowing the system to be easily used for scalability testing.

8 Related Work

Closed World Systems: The overriding characteristic of current provenance systems is the assumption of a closed world system – a contained environment over which the provenance system has full knowledge of all the data and processes used within the contained environment. Workflow-based systems such as [5, 27, 30] contain provenance for all of the executions and data that are executed by the system. Because the workflow is executed within this closed world system, complete provenance capture of the workflow run is possible. Moreover, since the provenance is used within the system, it can use the identity and storage based within those systems. In application-based provenance, certain applications are provenance capture enabled. For instance, in ES3 [15], or MapReduce [23], the applications used by scientists for data analysis are modified to capture provenance of their use. While these applications could be run over open, heterogeneous-style systems, they specifically create assumptions to form a closed world.

Identity: The MD5 hashing function was created in 1992 [24], while SHA-0 was developed by the National Security Administration (NSA) in 1993 and approved for use by the National Institute of Standards and Technology (NIST) in 2001 [22]. The work of [29] showed that the MD5 and SHA-1 algorithms were vulnerable to attack based on hash collisions. At present, SHA-256 is considered a secure algorithm.

Storage: Past efforts (e.g., [3]) have found relational databases to be of limited use, and achieved maximal performance once a native database for the given format was chosen; provenance, as a graph, is no different. Of interest, the tutorial guide for graph databases [25] cites provenance as an inherently good use of a graph database.

Testing: The provenance community has two styles of testing: actual generated provenance [1, 8, 16, 21] and the scalable but less empirical style presented in this work. As a community, we should be heading towards a benchmarking standard that tests query workload, use cases and scalability, just like the database community [2].

9 Conclusions

We have outlined some of the engineering decisions required to support a provenance system in an open world, one in which systems are not controlled or homogenous. New engineering designs are needed to support the real U.S. government applications we have observed. These systems tend to be less concerned with fine-grained and deeply detailed provenance, and more concerned with issues of maintaining graph connectivity, providing flexible and expansive query, and enabling capture in very heterogeneous environments with as little performance impact as possible. We describe solutions to identity, base storage, protection with utility, and scalability testing; all needed to make provenance a viable open-world solution. Our open-source provenance solution, PLUS, is at https://github.com/plus-provenance/plus.

Acknowledgements. The authors thank Len Seligman, Arnie Rosenthal, Maggie Lonergan, Paula Mutchler, Jared Mowery, Erin Noe-Payne, Zack Panitzke, Brenda Davies, Jesse Freeman, Blake Coe, and Sung Kim for their contributions to the PLUS system.

References

1. Provenance Challenge (2010). http://twiki.ipaw.info/bin/view/Challenge/
2. Transaction Processing Performance Council (2013). http://www.tpc.org/
3. Al-Khalifa, S., Yu, C., Jagadish, H.V.: Querying structured text in an XML database. In: SIGMOD (2003)
4. Allen, M.D., Chapman, A., Blaustein, B., Seligman, L.: Getting it together: enabling multi-organization provenance exchange. In: TaPP (2011)
5. Anand, M.K., Bowers, S., McPhillips, T., Ludascher, B.: Efficient provenance storage over nested data collections. In: EDBT, pp. 958–969 (2009)
6. Artem Chebotko, S.L., Fei, X., Fotouhi, F.: RDFPROV: a relational RDF store for querying and managing scientific workflow provenance. Data Knowl. Eng. **69**, 836–865 (2010)
7. Belhajjame, K., Gomez-Perez, J.M., Sahoo, S.: ProvBench (2013). https://sites.google.com/site/provbench/provbench-at-bigprov-13
8. Belhajjame, K., Zhao, J., Garijo, D., Garrido, A., Soiland-Reyes, S., Alper, P., Corcho, O.: A workflow PROV-corpus based on taverna and wings. In: Khalid Belhajjame, J.M.G.-P., Sahoo, S. (ed.) ProvBench (2013)
9. Benjelloun, O., Sarma, A.D., Halevy, A., Widom, J.: ULDBs: databases with uncertainty and lineage. In: VLDB, pp. 953–964, Seoul, Korea (2006)
10. Blaustein, B., Chapman, A., Seligman, L., Allen, M.D., Rosenthal, A.: Surrogate parenthood: protected and informative graphs. In: PVLDB (2010)
11. Chapman, A., Allen, M.D., Blaustein, B.: It's about the data: provenance as a tool for assessing data fitness. In: TaPP (2012)
12. Chapman, A., Blaustein, B.T., Seligman, L., Allen, M.D.: PLUS: a provenance manager for integrated information. In: IEEE Computer Information Reuse and Integration (2011)
13. Dey, M. Agun, M. Wang, Ludäscher, B., Bowers, S., Missier, P.: A provenance repository for storing and retrieving data lineage information. Technical Report, DataONE Provenance and Workflow Working Group (2011)
14. Foster, J.N., Green, T.J., Tannen, V.: Annotated XML: queries and provenance. In: PODS, pp. 271–280 (2008)
15. Frew, J., Metzger, D., Slaughter, P.: Automatic capture and reconstruction of computational provenance. Concurr. Comput. Pract. Exper. **20**, 485–496 (2008)
16. L. M. R. G. Jr., M. Wilde, Mattoso, M., Foster, I.: Provenance traces of the swift parallel scripting system. In: Khalid Belhajjame, J.Z., Gomez-Perez, J.M., Sahoo, S. (ed.) ProvBench (2013)
17. Mason, C.: Cryptographic Binding of Metadata. National Security Agency's Review of Emerging Technologies, vol. 18 (2009)
18. Missier, P., Chen, Z.: Extracting PROV provenance traces from Wikipedia history pages. In: EDBT (2013)
19. Missier, P., Embury, S.M., Greenwood, M., Preece, A., Jin, B.: Managing information quality in e-science: the qurator workbench. In: SIGMOD, pp. 1150–1152 (2007)
20. Moreau, L., Groth, P.: Provenance An Introduction to PROV. Morgan & Claypool Publishers, San Rafael (2013)
21. Moreau, L., Ludäscher, B., et al.: Special issue: the first provenance challenge. Concurr. Comput. Pract. Experience **20**, 409–418 (2008)

22. NIST, Descriptions of SHA-256, SHA-384, and SHA-512 (2001)
23. Park, H., Ikeda, R., Widom, J.: RAMP: A system for capturing and tracing provenance in MapReduce workflows. In: VLDB (2011)
24. Rivest, R.: The MD5 message-digest algorithm. IETF Working Memo (1992). http://tools.ietf.org/html/rfc1321
25. Robinson, I., Webber, J., Eifrem, E.: Graph Databases. O'Reilly Media Inc, Sebastopol (2013)
26. Rosenthal, A., Seligman, L., Chapman, A., Blaustein, B.: Scalable access controls for lineage. In: Theory and Practice of Provenance (2008)
27. Scheidegger, C.E., Vo, H.T., Koop, D., Freire, J., Silva, C.: Querying and re-using workflows with VisTrails. In: SIGMOD (2008)
28. W3C, Provenance Data Model (2013). http://www.w3.org/TR/prov-dm/
29. Wang, X., Yu, H.: How to break MD5 and other hash functions. In: Cramer, R. (ed.) EUROCRYPT 2005. LNCS, vol. 3494, pp. 19–35. Springer, Heidelberg (2005)
30. Wolstencroft, K., Haines, R., et al.: The taverna workflow suite: designing and executing workflows of web services on the desktop, web or in the cloud. Nucleic Acids Res. **41**, w557–w561 (2013)
31. Xiey, Y., Muniswamy-Reddy, K.-K., Fengy, D., Liz, Y., Longz, D.D.E., Tany, Z., Chen, L.: A hybrid approach for efficient provenance storage. In: CIKM (2012)