School of Informatics

Informatics Research Review
Configuration of Provenance Metadata in Relational Databases

Exam No. B132486
August 2019

Abstract

Data derivation from analysis of source data (data collected and stored in databases) is continually used to draw inferences and form the underlying foundation for research. Questioning the authenticity and validity of the derived data is a plausible argument taking into consideration the potential ripple effect that the use of derived data may have. Relational databases based on SQL have been hailed for their efficiency in computation, but there is an evident disconnect between query efficiency and traceability mechanisms implemented in the standard. This review delves into mechanisms and techniques explored to warrant traceability of derived data.

Signature: EMM Date: 14th January, 2019

Supervisor: Iris Kyranou

Contents

1 Introduction

Data collection, aggregation and analysis have rightly been the core of research especially in the advent of the internet of things and advanced (big)data systems. This has resulted in creation of enormous datasets; some from research data, others generated from analysis of research data and some derived from taking subsets of the original data. This data is used for analysis and oftentimes is a basis upon which critical decisions are made. Additionally, it is adopted for deep learning systems [1] to find patterns, draw inferences and guide research.

Setting up a protocol by which to establish provenance of not only the derived datasets as aggregates but also that for more concise constituents like relations, attributes and tuples is of prime importance. Where-provenance and why-provenance are data traceability terminologies describing locations of the source database from which the derived data was extracted and why parts of the source data influenced the result of the derived data set. The complexity of establishing this lineage is magnified by existence of vague information, like null values, in databases using the Structured Query Language (SQL).

Although SQL queries running on complete data sets return accurate data, it has been shown that querying incomplete relations often returns inaccurate and/or unexpected data. This is the principal source of null values. Nulls are ambiguous, as they can represent many forms of data types and their evaluation from queries in SQL yields ambivalent data. Among the approaches suggested to deal with incomplete information evaluation in SQL is a modification of the standard itself, to provide the possibility of computing certain answers and/or possible answers depending on the context of the intended query. Finding these certain answers is a complex problem falling under the co-NP complexity class [2] because SQL is based on relational calculus and relational algebra.

Solving this problem is not a solution to proving validity of data but is a solution for finding accurate data when querying incomplete data sets. If a mechanism to accurately prove data lineage is adopted, it is improbable that ambiguity caused by queries executed on incomplete data arises because it would be possible to trace sources of incompleteness and debug before querying on incomplete data. This infers that provenance solves both the ambiguity of incomplete derived datasets as well as justifying their existence.

The challenge escalates when debugging distributed systems because collaboration and interactions among nodes makes traceability of generated data more complicated. Part of this complexity arises from load balancers randomly distributing tasks among live servers. This makes it evident that justifying validity of a dataset or debugging it by only analyzing the data by reverse data management algorithms is unlikely to succeed.

This review looks into progress that has been made in data provenance and ongoing research in this faction of databases and data management with a focus on relational databases using the standard SQL. The intention is to highlight the importance of data traceability, how to implement mechanisms that capture filiation metadata, what to capture for a comprehensive track back of data parentage and techniques that could be adopted to store and represent provenance metadata to aid its dissemination.

2 Literature Review

2.1 Understanding Provenance

Suppose a database view V from a query Q on database D; V = Q(D), in practice,

```
SELECT name
FROM Customer
WHERE age > (SELECT AVG(age)
             FROM Customer);
```

The names in the output are a result of the average of all the values in the attribute age of relation customer. An alteration on any of the values contained in attribute age could potentially affect the result of this query. This is the why-provenance, a description of reasons that contribute to the output containing the data that it does. Where provenance, in comparison, traces sources of data that combine to generate tuples of the output. A combination of why and where provenance is necessary to debug errors in a derived data set. In the above query, if the output contains unexpected results, the use of provenance methods to debug elucidates the reason(s) and the source(s) of data in the queried database that lead to some output.

Cheah et al. [3] discuss parameters that ought to be documented when capturing provenance; the time of capture, correctness and completeness of provenance metadata, queries run on provenance metadata and their storage. These parameters allude that provenance elucidation is a procedural operation, particularly aided by the time of capture parameter that guides on the chronology of data transformation. In contrast with forward-moving data flows that most research on databases is encompassed on, Reverse Data Management (RDM) [4] has been studied, where action needs to be performed on certain inputs in order to achieve some specific output data. since data transformation takes this forward-moving paradigm for a variety of functionalities like query processing, data integration, data mining, clustering and indexing, RDM models are be useful for data cleaning, causality computation and data re-generation. Provenance is a RDM model.

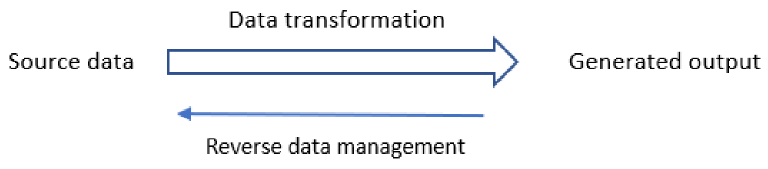


Figure 1: Reverse data management[5]

Data causality as a refinement of provenance investigates why an output deviates from the expected result and rolls back to select the appropriate tuples that may lead to a desired output, as a part of curation. Alexandra et al.[5] describe a form of How-to queries, that take a hypothetical stance on the state of a pair of source and target data by asking questions amounting to how a tweak of the source data reciprocates in changes in the output. The challenge in implementation of the how-to queries lies in a requirement to extend SQL in order to maintain the declarativity and ensure efficient evaluation.

Since SQL's core is relational algebra, this requirement is needless in relational database, as first shown by Codd in 1971 [6]. Codd provides an algorithm "for translating an arbitrary alpha expression into semantically equivalent sequence of operations in relational algebra." The algorithm demonstrates that relational algebra has at least the selective power of relational calculus. Considering that the how-to queries are selective queries, extension of the SQL standard for a specific fragment of selection queries may be regarded as overkill. Additionally, Guargliardo in 2017 [7] argues that it is a fundamental result of relational database theory that the expressiveness of the basic declarative query language, calculus, is the same as that of the basic procedural query language." If an addition to SQL's calculus is enacted, there needs to be mathematical proof of equivalence in the translation between relational calculus and relational algebra for purposes optimization and translations between the two. This has not been extensively explored by Alexandra [5] and should form a critical part of consideration.

2.2 Provenance Models

2.2.1 Annotation

Appending metadata to every cell of every tuple, resembling l-values that point to a storage location [8] in programming has been suggested as a possible model. The appended data is a path to the location of source data. A deterministic tree is formed with the pair x:y. X is an edge label and y being the node containing data. Keys are used as edge values for relational databases as is in figure 2. In object-oriented database systems, persistence of objects is utilized as the label to nodes. This approach cannot be directly adopted to relational databases due to the lack of objects.

Buneman et al.[9] prove that "[in general] where-provenance is not invariant over the space of equivalent queries and that a purely syntactic characterization of where-provenance is unlikely

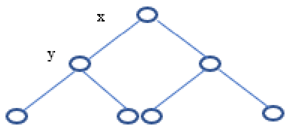


Figure 2: A deterministic tree.

to yield a complete description of the where-provenance." However, the use of a restrictive faction of queries enable traceability of queries for where-provenance. The trade-off between restricting the expressivity of queries in SQL for the ability to trace origins is not convenient for the standard as general progression of the expressive power of SQL is increased, not restricted. Buneman's described system of rewrite rules preserves why-provenance over the class of "*well-defined queries*" whereas traceable queries preserve where-provenance. He suggests that future work should be done to define a set of conditions for the class of well-defined queries and that additional functional dependencies should be explored to help in better description of the where-provenance. Application of functional dependencies to preserve where-provenance is more applicable that restricting the constructs of the language because functional dependencies enact restrictions only when they are triggered, as first shown by Codd in 1972 [10].

Simmhan et al.[11] provide an alternative to restriction of queries by conceptualizing data products and their transformations as a Directed Acyclic Graph(DAG) [12].

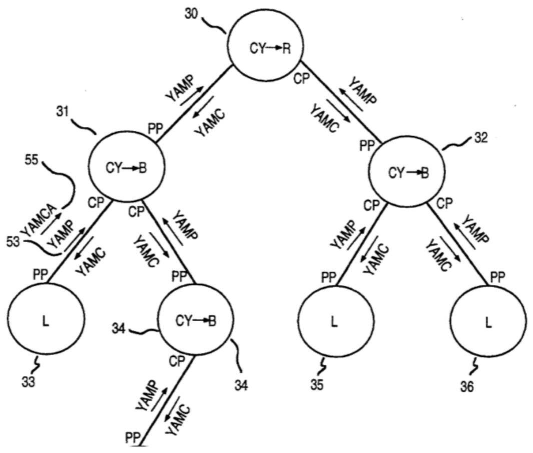


Figure 3: A sample directed acyclic graph[12]

Capturing data filiation in this case would have edge labels representing transformation processes while nodes would be the output data products. The implementation would be in the workflow engines. However, data transformations generated automatically by workflow engines in runtime (dynamic workflows) may be inaccurate in provenance generation because of the non-deterministic dynamic environment as opposed to static workflow engines whose transformation traces are generated automatically and later annotated by users because their runtime environment is pre-determined.

In the wake of software development increasingly getting intertwined with artificial intelligence, the runtime environment of computer programs certainly is more dynamic than ever before. This revelation forces workflow engines that implement provenance capture for dynamic runtime environments. If workflow engines are unsuccessful in creating these traces, then service providers and clients have the onus to generate interaction logs. This is not a reliable mode of recording data genealogy because in a distributed system, some clients or servers may be compromised to exhibit byzantine behaviour as discussed by Paul[13]. Even by assuming compliance of all client nodes to execute as specified in any given protocol, the complexity of recollecting data flows from distributed systems is only best suited for shell interfaces where the command line provides mechanisms for logging input data, commands and yielded outputs. Rendering provenance information from this is then straightforward, given the level of detail included in these logs. This panoramic view of transformation guarantees accurate and complete traceability of every data item from inception to its current state, as discussed by Simmhan[11]. This level of detail included in provenance capture is directly proportional to the richness of provenance metadata. Despite the criticality necessitating provenance capture, it is important to avoid cluttering this metadata with too much detail, which might result to new challenges like storage and retrievability efficiency as well as incomplete provenance metadata capture.

2.2.2 Provenance in Distributed Systems

Debugging and provenance methods become more challenging across distributed systems as Whittaker et al. [14] elaborate. The distinction between causality and why-provenance of data on distributed systems is highlighted. Causality is a formalism specifying all possible events that may have led to the generation of a tuple or relation. This is approach is very general as it includes noisy events; those that may have resulted to the generation of tuples that did not influence the environment that resulted in the output of interest. Why provenance on the other hand assumes a static mode of databases, incapacitating it from ably handling the time-varying nature of distributed systems. In highlighting these inadequacies of both models (with causality lacking data dependence and why provenance lacking liveness) the paper proposes a wat-provenance (why-across-time) combining properties of the two approaches. Considering a Postgres database that stores data from distributed servers, a sequence of events might lead to data flow between the servers to by-pass application level restrictions. The paper describes how a social media application might display information to users after their privileges of accessing this data have been revoked as a result of caching by servers.

In a scenario where Bob and Ava are social media users of a Facebook-like network, if Ava revokes the right of Bob from accessing contents that she puts up on the network, the load-balancer forwards this request to Redis-Backend server s1. Following this request, Ava posts new content, which the load-balancer forwards to server s2. Server s3 then issues a request to pull data to its cache, which includes data from Avas latest content. If Bob refreshes his view and the load-balancer assigns his request to server s3, he will have accessed Avas content. Bobs access to content from Ava cannot be restricted by application level filters. In this respect, debugging this scenario using the relational databases why-provenance may prove helpful only at the Postgres database. Examining the database log files will show events in this order; Ava revoking Bobs access to her content, Ava creating new content and server s3 sending a query to fetch data. Debugging this situation using causality would result in checking every message through server s3, even those not involving Ava and Bob, an over approximation. This analysis requires wat-provenance, a combination of both why-provenance and causality. Wat-

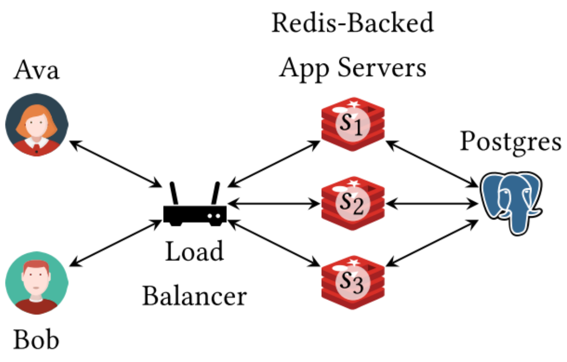


Figure 4: Client-server interaction in a distributed system [14].

provenance is only applicable in a deterministic environment. If the load-balancer randomly forwards requests to any of the three servers, then causality, which is a part of wat-provenance might take a very large active domain because all messages sent to all three servers prior to Bobs request would have to be examined.

2.2.3 Provenance From Log Files

Capturing provenance by program instrumentation (which is tweaking programs/applications to append metadata about lineage) requires data scientists to access source code of all contributory programs that give rise to the dataset being analyzed. Ghoshal et al. [15] highlight two classes of provenance, process and data provenance. Process provenance is the description of a process that triggered provenance capture by storing metadata about the process and the data related to the process execution. Data provenance is metadata associated with the derivation history of data items involved in execution. A further classification of provenance surfaces, prospective and retrospective provenance. Prospective provenance involves the capture of an execution specification, i.e. the steps that should be followed to process and generate data. Retrospective provenance, on the other hand, highlights actual steps taken during execution as well as the execution environment during these steps. Experimental results show that log files contain fairly large pieces of information from which provenance can be derived. As shown in figure 5, it also highlights that sizes of log files do not correspond linearly to the amount of provenance information that can be derived from them, but provenance corresponds to the succinctness of rules that derive provenance-aware information and on the semantic accuracy of logs.

This illustrates than it is impossible to derive any provenance information without logs. A technique to capture provenance-aware information can be drawn from Nakamoto's [16] solution to the double spending problem by building a p2p distributed timestamp server to generate computational proof of chronology of events. In the case of relational databases, server logs already implement the log before write in SQL.The execution environment should sufficiently provide proof for any derived data. This environment consisting of variable states at the point of data generation and commands being executed by the program running queries on the data can wholly express provenance of output data from the environment. Bitcoin uses an ownership traceability mechanism for every coin from the point of creation onwards by attaching the

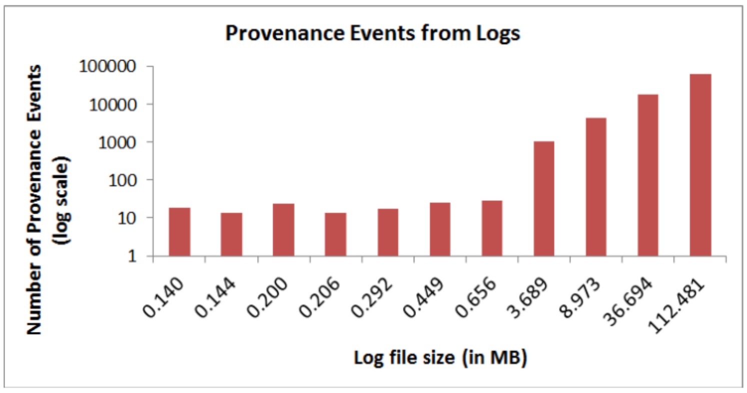
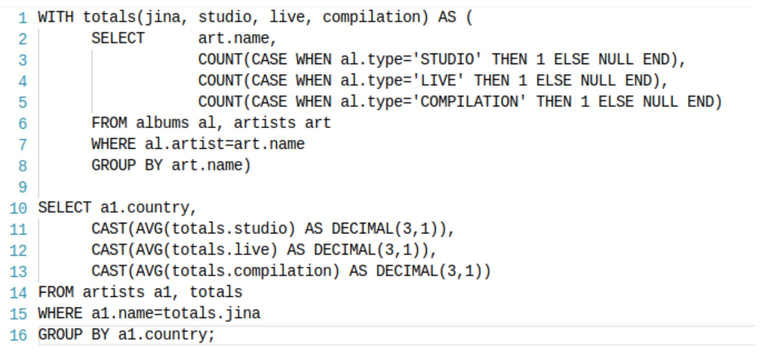


Figure 5: Provenance information in log files [15].

current owners digital signature, a hash of the previous transaction and the public key of the next owner to the end of coin information. A payee intending to verify proof of coin ownership would verify attached signatures from every previous owner. In the context of SQL, coin ownership could be loosely juxtaposed as data written into storage and intermediary processes during execution that hold data at some point in main memory (such as sub-queries). Borrowing from bitcoin traceability, Satoshis implementation of proof of ownership by adding digital signatures at the end of every coin information can be mapped to SQL by adding extra bits to every data generation. This might be implementable, but consideration needs to be taken for intermediary data. As is in the figure 6, the first 8 lines of the query present transitional data that resides in main memory during execution and is to be used as input data for the main query. If the strategy were to append extra bits of metadata to every byte of data as the carrier to environment state and the query executed, provenance may have data incompleteness due to these sub-queries, since their states are only temporary and would not be written into storage unless this transitional information is included into the final output data.



```
 1 WITH totals(jina, studio, live, compilation) AS (
 2       SELECT      art.name,
 3                   COUNT(CASE WHEN al.type='STUDIO' THEN 1 ELSE NULL END),
 4                   COUNT(CASE WHEN al.type='LIVE' THEN 1 ELSE NULL END),
 5                   COUNT(CASE WHEN al.type='COMPILATION' THEN 1 ELSE NULL END)
 6       FROM albums al, artists art
 7       WHERE al.artist=art.name
 8       GROUP BY art.name)
 9
10 SELECT a1.country,
11       CAST(AVG(totals.studio) AS DECIMAL(3,1)),
12       CAST(AVG(totals.live) AS DECIMAL(3,1)),
13       CAST(AVG(totals.compilation) AS DECIMAL(3,1))
14 FROM artists a1, totals
15 WHERE a1.name=totals.jina
16 GROUP BY a1.country;
```

Figure 6: Query with placeholder data

In addition, for distributed systems, there needs to be a mechanism in place for consolidating logs from different modules in a chronological/logical order and linking information contained therein from which provenance should be derived. For this purpose, a unique context ID for each execution instance of the application could be assigned. Alternatively, batching log files for every execution singularly. The paper highlights that a trade-off must be struck for either of these approaches. Context IDs require tweaking source code of logging mechanisms whereas batching logs adds to the complexity of scheduling.

To be noted is that since provenance is a measure of data quality, it should be possible to quantify the quality of provenance itself. Ghoshal [15] highlights that for large log files, a rule-based scheme should be employed to filter out the relevant logs, those that contain provenance worthy information. Of the desiderata highlighted by the paper by Ghoshal [15] for quantifying quality of provenance, redundancy and timestamp are of interest. Redundancy implies that logs that have duplicate information lower the quality of provenance as they increase the amount of data to be analyzed. Timestamps may provide a way of extracting provenance from distributed systems.

2.3 Methods of Provenance Capture

Simmhan [11] compares inversion queries and annotation as the main methods of automatic provenance capture. He purports that formal techniques to invert some relational queries exist, but they are restricted to a certain class of relational queries and are not universally applicable. An advantage of the inversion technique is to represent a whole view/dataset in a single inversion query, synoptically answering the why-provenance question. Annotation, in contrast, could include metadata containing lineage information comprehensively answering the where-provenance about every cell of every tuple. This posses a tradeoff; with annotation offering more provenance-rich metadata hence higher quality of assurance and inversion queries compactly expressing all cells of an attribute for the whole relation. From this review, it is evident that to completely capture the execution environment there is need to capture both the transformation process as well as the origins that give rise to output datasets.

Consider a query like that portrayed in section 2.1. If an audit trail of view V were to be followed through, reverse engineering of data lineage by inversion queries can only elucidate inclusion of names in the output. If the input dataset cannot be availed, it is impossible to provide the omitted names only from data in the output, suggesting that all the traces of input datasets ought to be availed for such a trace. Annotation to every cell of every tuple can accurately describe provenance but as highlighted in the paragraph above, significantly impedes the speed and efficiency of query execution in SQL. Similarly, appending such metadata to output views V sourcing their inputs from multiple source tables containing considerably large amounts of data might consume large chunks of memory and hamper execution speed. In fact, the rate at which provenance metadata can potentially grow should be put into consideration, as detailed provenance information may at some point grow to consume larger memory than the data after serial transformations. This cost needs to be reviewed because it is probable that this provenance information would not be used after all. Additionally, annotation of source data to the output might be viewed as an unnecessary replication of data.

Widom [17] proposed a database system to manage data accurately whilst preserving information guaranteeing lineage traceability as an understandable extension to SQL. She presents a prototype of the Trio Data Model (TDM) that captures data, accuracy and lineage information. The TDM prototype offers lineage data for every tuple. If a tuple changes (updates and deletions), "new data values are inserted into the database while old data values are "expired" but remain accessible in the system". This implements the possibility of several forks for a tuple as data is not overwritten, synonymous with Nakamotos [16] bitcoin implementation where the distributed ledger may fork due to divergent data contained in generated blocks.

The system implements querying of lineage data i.e. "find all records whose derivation includes data from relation R," or "determine whether a particular record r was derived from any data imported on 4/1/04." and attempts to answer a provenance model not previously explored, "*when-provenance*", as Widom argues the time of inception contributes to increasing confidence in parentage traceability. She also highlights the ability to use lineage information to as a guide for data modification and update should there be a change in ancestral data.

3 Summary & Conclusion

The question of why and how to capture provenance arises: Are data transformations or the procedural transformation process solely sufficient to describe provenance or is there need to capture both in order to accurately describe a datasets lineage? The DAG as described in the section 2.2.1 captures both the transformation and data involved. This suggests that to comprehensively document the execution environment there is need to record both the transformation process as well as the origins of data that yield the output datasets. As discussed in section 2.1, ascertaining provenance can be viewed as a procedural operation and would be best captured during parsing in Database Management Systems (DBMS) as discussed by Popescu et al. [18] on composing statistical parsing with semantic tractability. Further, it is evident that time of events is a prime component for tracing provenance in distributed systems, otherwise the cyclic interactions among multiple nodes would complicate finding parentage. It is also critical to explore whether backing up of provenance metadata is of any importance and if so, how and when in the data generation/transformation process it should be effected.

Future research should consider the implementation of combining both annotation and inversion queries (assuming availability of datasets from inception) which comprehensively describe the evolution of data by appending only 2 pieces of metadata; pointers to all contributing tuples and the transformation processes. To uniquely identify tuples, each should be assigned an immutable identifier upon inception (whether the data is collected from user input or generated from analysis). Pointers are to use these IDs to reference sources of transformations. Research should also be aimed at contriving a standard for metadata capture that would facilitate universal readability and dissemination across different platforms, as SQL has been.

References

[1] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *nature*, 521(7553):436, 2015.

[2] David S. Johnson. The np-completeness column: an ongoing guide. *J. algorithms*, 6(3):434–451, 1985.

[3] You-Wei Cheah and Beth Plale. Provenance analysis: Towards quality provenance. In *E-Science (e-Science), 2012 IEEE 8th International Conference on*, pages 1–8. IEEE, 2012.

[4] Ang Chen, Yang Wu, Andreas Haeberlen, Boon Thau Loo, and Wenchao Zhou. Data provenance at internet scale: Architecture, experiences, and the road ahead. In *CIDR*, 2017.

[5] Alexandra Meliou, Wolfgang Gatterbauer, and Dan Suciu. Reverse data management. *Proceedings of the VLDB Endowment*, 4(12), 2011.

[6] Edgar F Codd. A data base sublanguage founded on the relational calculus. In *Proceedings of the 1971 ACM SIGFIDET (now SIGMOD) Workshop on Data Description, Access and Control*, pages 35–68. ACM, 1971.

[7] Paolo Guagliardo and Leonid Libkin. A formal semantics of sql queries, its validation, and applications. *Proceedings of the VLDB Endowment*, 11(1):27–39, 2017.

[8] Rene Brun and Fons Rademakers. Rootan object oriented data analysis framework. *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment*, 389(1-2):81–86, 1997.

[9] Peter Buneman, Sanjeev Khanna, and Tan Wang-Chiew. Why and where: A characterization of data provenance. In *International conference on database theory*, pages 316–330. Springer, 2001.

[10] Edgar F Codd. Further normalization of the data base relational model. *Data base systems*, pages 33–64, 1972.

[11] Yogesh L Simmhan, Beth Plale, and Dennis Gannon. A survey of data provenance techniques. *Computer Science Department, Indiana University, Bloomington IN*, 47405:69, 2005.

[12] Florin Oprescu. Method and apparatus for unique address assignment, node self-identification and topology mapping for a directed acyclic graph, February 28 1995. US Patent 5,394,556.

[13] Paul Brutch and Calvin Ko. Challenges in intrusion detection for wireless ad-hoc networks. In *null*, page 368. IEEE, 2003.

[14] Michael Whittaker, Cristina Teodoropol, Peter Alvaro, and Joseph M Hellerstein. Debugging distributed systems with why-across-time provenance. In *Proceedings of the ACM Symposium on Cloud Computing*, pages 333–346. ACM, 2018.

[15] Devarshi Ghoshal and Beth Plale. Provenance from log files: a bigdata problem. In *Proceedings of the Joint EDBT/ICDT 2013 Workshops*, pages 290–297. ACM, 2013.

[16] Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system. 2008.

[17] Jennifer Widom. Trio: A system for integrated management of data, accuracy, and lineage. Technical report, Stanford InfoLab, 2004.

[18] Ana-Maria Popescu, Alex Armanasu, Oren Etzioni, David Ko, and Alexander Yates. Modern natural language interfaces to databases: Composing statistical parsing with semantic tractability. In *Proceedings of the 20th international conference on Computational Linguistics*, page 141. Association for Computational Linguistics, 2004.