Scaling SPADE to "Big Provenance"

Ashish Gehani Hasanat Kazmi * Hassaan Irshad

SRI International

{ashish.gehani,hasanat.kazmi,hassaan.irshad}@sri.com

Abstract

Provenance middleware (such as SPADE) lets individuals and applications use a common framework for reporting, storing, and querying records that characterize the history of computational processes and resulting data artifacts. Previous efforts have addressed a range of issues, from instrumentation techniques to applications in the domains of scientific reproducibility and data security. Here we report on our experience adapting SPADE to handle large provenance data sets. In particular, we describe two motivating case studies, several challenges that arose from managing provenance at scale, and our approach to address each concern.

1. Introduction

SPADE [26] is SRI's open source data provenance middleware. The design of the first generation hindered scalability in several respects. The collection and storage of provenance records were tightly coupled to ensure that the metadata was always synchronized with the data that it described. In particular, it used file system call interposition to track changes in persistent data, recording the results in a relational database. The throughput of workloads with bursts of file system activity was limited by the transaction rate supported by the backing store. Similarly, query performance was limited by the type of database used. Further, the components were statically coupled. This slowed the system's use in new domains since all the components had to be modified to accommodate changes in the implicit schema. Every provenance record was digitally signed and all query responses were cryptographically verified. This imposed runtime overhead from added computation. Provenance records were propagated along with data flows in distributed settings, introducing latency and storage overhead.

The architecture of the second generation was significantly revised to address the above issues (and others [10]). A provenance *kernel* was introduced to provide a system service that mediates the collection, storage, and querying of provenance records. Internal buffering in the kernel allowed provenance *reporters* to generate bursts of activity at higher rates than configured provenance *storages* could support. Decoupling the storage allowed graph, relational, or other database types to be used, depending on the expected workload and query characteristics. The kernel exposed uniform, extensible interfaces for reporting and querying provenance. This allowed reporters for new domains to be developed rapidly. Cryptographic signing and verification were factored out, and can be added back when needed. Provenance collection is decentralized, reducing storage overhead from replication. Distributed query performance is accelerated through the use of *sketches* [8].

The provenance kernel is agnostic to the domain from which activity is reported. It exposes an interface that allows provenance elements to be reported, initially using the Open Provenance Model (OPM) [20] and more recently the W3C PROV [23] data model. Apart from this, no constraints are placed on the semantics of the elements being ingested. Each reporter is free to define independent sets of attributes about the provenance that it collects. This has allowed the development of multiple reporters that exploit different instrumentation techniques to provide dissimilar properties. Once the provenance elements reach the kernel, they are transformed into and treated as generic graph vertices or edges. While this affords the storage subsystems extreme flexibility in their management of the provenance records, it introduces a challenge for managing "big provenance", as we describe further in Section 3.

In the course of multiplexing streams of provenance from multiple reporters, the SPADE kernel can apply a set of provenance *filters* before demultiplexing the streams into each of the configured storages. Each filter can make arbitrary changes to incoming provenance vertices and edges. For example, OPM elements can be converted into a PROV embedding. Filters can be composed arbitrarily. This property makes them useful for integrating provenance [9]. In particular, filters have been used for aggregation of temporally-related events, fusion of streams with partially overlapping elements, and combining provenance from different abstraction layers. However, filters are designed for stream processing, necessitating the use of finite windows

* While visiting SRI.

TaPP 2016, June 8–9, 2016, Washington, DC.

within which candidates for integration must arrive. This proved to be a significant limitation in practice for the case studies that we describe in Section 2.

Section 3 describes three challenges that we faced with large provenance data sets. First, incomplete knowledge about how the information will be used and the high cost of collection requires that all details must be retained. Second, a stream processing approach for data integration inherently trades memory usage (for tracking a longer history) with lost integration opportunities (when relevant events are not observed within a temporal window). Third, to ensure that the persistent representation of the graph remains connected, independent reports about the same vertices must be reconciled. If persistent storage is consulted, this can significantly reduce the rate at which provenance can be ingested.

Our approaches to address the above concerns are described in Section 4 and evaluated in Appendix A. We conclude in Section 5. Related work is outlined in Appendix B.

2. Case Studies

SPADE has been utilized in a variety of contexts. In one of its earliest uses, the provenance records were leveraged to identify the files needed when staging software releases. Other examples of its use include minimizing the re-execution of workflows [19], performing fault diagnoses [15], and identifying privacy-sensitive data flows [29]. In each case, SPADE has been deployed as a *data microscope* – that is, it has been configured with a narrow focus. In particular, the instrumentation would collect a predefined subset of provenance information necessary to answer a particular set of questions. Further, provenance metadata was only recorded for the duration of a single analysis or experiment each time.

In contrast, the two case studies we describe below required large volumes of provenance metadata to be collected for extended periods of time to facilitate the range of analyses that were of interest. The expanded volumes and timescales highlighted architectural limitations, as we outline in Section 3, that led to design changes and new functionality, as described in Section 4. It is worth noting that despite SPADE being designed for distributed settings, it is used as a centralized service on a single host in both of the below settings.

2.1 Bitcoin

Bitcoin was proposed in 2008 [21] as the first completely decentralized cryptocurrency. It supports anonymous users, has no central mint, and distributes the verification effort needed to ensure that participants do not spend their cash twice. The Bitcoin market size crossed $1 billion [6] in 2013. 75,000 mainstream companies, from Dell to Expedia, accepted payment in bitcoins by 2014 [18]. However, it was also heavily used in black markets, as could be seen from crawls of the "dark web" [5] (typically accessible only via Tor hidden services).

Bitcoin can be viewed as a history of transactions, with its public ledger (known as the *blockchain*) reporting the provenance of all payments sent and received in the ecosystem. Ingesting these records into SPADE enables a wide range of provenance-based queries. In particular, this allows the discovery of all paths (even through multiple intermediate transactions) between a Bitcoin payer and payee; an ancestral lineage query returns all payers whose bitcoin found its way to a specific transaction or payee; a query for descendants determines all payees who received all or part of a payment; and, inspection of an agent vertex allows all their incoming and outgoing payments to be directly identified. Without importing Bitcoin into a provenance framework, such queries would require the traversal of the entire blockchain. While the use of a provenance framework provides optimized support for such queries, it must address the challenge of managing large provenance graphs. Even the first 200,000 blocks of the Bitcoin blockchain, when there was relatively little activity, results in 31 million vertices and 55 million edges. After 398,000 blocks (which is 99% of the current blockchain) have been ingested into SPADE, the graph database contains 522 million vertices and 1,042 million edges. With indexes, it takes 468 GB of disk space.

2.2 Forensics

"Common Criteria" is an abbreviation for an ISO standard [17] used to certify the security of computers. It was created in 2005 to unify earlier U.S., Canadian, and European standards. The Common Criteria provides a framework for users to define their security requirements, vendors to identify the features of their products, and testing laboratories to evaluate the claims. Users range from governments to companies managing critical infrastructure. Even in environments where individuals are not assumed to be hostile, monitoring is a significant requirement [4]. Linux's Audit framework is designed to comply with this stipulation. Its use is an integral part of securing a system, as detailed in the guidelines from vendors of enterprise distributions, such as RedHat [25], Oracle, [22], and SUSE [27].

If a computer with an audit trail is involved in a security violation, an investigator will use the records to reason about the sources, methods, and consequences of the attack. This involves connecting the events reported into a timeline that explains what transpired. The process can be automated if the audit logs can be ingested into a provenance framework, such as SPADE [10]. In particular, provenance queries allow a forensic analyst to rapidly identify the history of activity in the system that led to the creation of particular anomalous data artifacts, as well conduct impact analyses to determine the set of processes and files that were affected by a breach. However, even a moderately loaded web server generated provenance records at the rate of 210 million vertices and 833 million edges a day.

3. Challenges

After we started using SPADE to manage larger provenance datasets, we encountered the issues described below.

3.1 Collection / Querying

Gathering information about the agents, activities, and artifacts in a system (and the transformations that they undergo) depends on deploying instrumentation or drawing inferences from extant data. When the end goal is defined, it is possible to focus the choice of *coverage*, *abstraction level*, and *temporal granularity*. For example, if the provenance will be used to diagnose where a scientific workload's I/O hotspots lie [11], coverage of unrelated activity in the system can be excluded; if the goal is to identify privacy-sensitive information flows in a mobile application [29], provenance at the abstraction level of function call invocations suffices; if the information will guide the staging of software releases [10], simple dependency analysis is required without fine-grained temporal access records.

Collecting a large dataset requires a significant investment, precluding the repetition of the operation each time a different variant of attributes is needed. This motivated by several factors, including the length of time it takes to gather the information, the computational resources to process the records, and the investment in storage needed. Consequently, it becomes necessary to utilize instrumentation that provides maximal coverage, refined abstractions, and high temporal fidelity. However, this creates a challenge when querying the provenance, since the responses will include extraneous details that are not of interest.

3.2 Integration

The history of a computation is modeled as a typed, annotated graph of provenance nodes and relations. SPADE supports the use of sequences of filters that operate on the resulting stream of graph elements. This allows operations such as the *aggregation* of temporally-related records, as well as the *fusion* of streams that describe different aspects of the same underlying phenomenon.

Integration was previously effected in filters by combining multiple incoming provenance elements into fewer outgoing ones [9]. In order to do this, each filter cached elements that arrived earlier in the stream until the later ones arrived. However, this design implicitly assumed at least loose temporal synchronization between provenance elements that were candidates for integration. When an experimental analysis was conducted in a limited window of time, this assumption always held. However, when provenance is collected over long spans of time, the unbounded growth of the cache presents a challenge (due to the adverse impact memory pressure has on system performance).

3.3 Storage

SPADE utilizes a stream processing paradigm. Provenance vertices and edges are moved from reporters to kernel buffers, have filters applied to them, and are committed to persistent storage as soon as possible. The advantage of this approach is that it minimizes the buildup of intermediate state at each point in the system. This facilitates scaling with large provenance datasets.

The same provenance element may need to be defined multiple times. For example, this is the case for a vertex that represents an operating system process. Each time the process reads a file and a new edge is constructed to report the event, one endpoint will consist of the same vertex. The provenance storage can reconcile multiple reports of the same provenance element if it includes a unique identifier. Prior to storing a new provenance element, the storage can check to see if it has previously been stored. However, this introduces a challenge to scaling since such queries are expensive and limit the rate at which new provenance can be stored.

4. Scaling

To address the challenges outlined in Section 3, we extended SPADE's architecture, changed the design of components, and optimized the implementation in several ways.

4.1 Transformers

The streams of provenance being reported are collected in pools of persistent storage. The most direct method for analyzing the resulting forest of provenance graphs is to utilize the indexing and query language of the backing database. However, such systems do not support provenance-specific operations directly. Consequently, the SPADE kernel exposes a query interface, to which clients can send such requests. Each provenance query is translated into a sequence of queries in the language of the underlying storage. At the end of the process, which may be iterative or recursive, the resulting provenance graph is returned to the client.

Section 3.1 described the tension between collecting the most detail available versus application-specific provenance utilization at coarser abstraction levels. This motivated the introduction of *transformers*, a fifth point of extensibility in SPADE's architecture (after those offered by the existing *reporter*, *filter*, *storage*, and *sketch* interfaces). A transformer operates on the graph that results from a provenance query (as illustrated in Figure 1). Multiple transformers can be configured, which allows their operations to be composed. Each transformer can make arbitrary changes to the graph it receives, after which it sends the modified graph to the next transformer. After the last has made its changes, the result is returned to the querying client.

Transformers can serve as lenses that focus on details at a particular level while excluding or abstracting other information. In particular, if the provenance dataset is used for a specific application, the installation of appropriate transformers allows the queries to be framed in a corresponding data model. We provide examples in Section A.1.

4.2 Content-Based Integration

Most storage systems decouple the namespace of identifiers used to refer to pieces of content. For example, the content stored in a file is independent of its name and location in typical filesystems. Similarly, the contents of a cell in a relational database are independent of the column and row in which it is located. The advantage of this approach is that references to the data container do not need to be updated when the contents change.

In an alternative approach, the identifier used to refer to a piece of data is derived directly from it. In typical *content-based storage*, the identifier is a cryptographic hash of the data stored in a file. We adapted SPADE's graph database storage to utilize this paradigm (with SHA-256 as the hash). When a provenance vertex arrives for storage, a check is performed for the existence of an instance with the same set of annotations. The vertex is only stored if no match is found. When an edge arrives, if an endpoint vertex matches an extant instance in the storage, the edge's endpoint is changed to the version already in the persistent store.

A drawback of this design is that independent provenance graphs that contain vertices with the same annotations will become connected through the merged vertices. However, this is easily addressed by adding an extra annotation to indicate which graph each vertex belongs to. This step ensures that provenance forests are not merged into single connected graphs.

Such content-based storage provides a powerful primitive for performing provenance integration. Each monitored phenomenon gives rise to a set of domain-specific annotations. These decorate the provenance graph vertices and edges. We term a provenance element *content-based integratable* if all its annotations will match every time two instances need to be merged. (For example, an agent may control multiple processes. This may result in a separate agent description accompanying the reporting of each process. If the content of multiple agent vertices match, the vertices can be integrated into a single vertex.) Similarly, we term an application domain suitable for *content-based integration* if it can be mapped to a schema where all desynchronized provenance elements that need to be merged are content-based integratable.

This strategy does not support the full range of data integration functionality that is possible. However, it is sufficient to allow integration for some large provenance datasets, including the two cases we studied.

4.3 Storage Screening

Providing provenance elements with unique identifiers allows multiple reports of the same element to be reconciled in the storage subsystem. As Section 3.3 explains, this introduces performance overhead. In a baseline design, the slowdown derives from the fact that a query is made to the underlying storage each time an element arrives. In an optimized design, a cache is maintained to screen reuses of the same provenance element. The cache size is grown dynamically to utilize all available memory. However, the increased memory pressure creates an alternate source of overhead.

The final design utilizes a hybrid approach, with a Bloom filter as the primary screen and a cache as a secondary screen. The cache has fixed size with a first-in-first-out (FIFO) eviction policy. When a provenance vertex arrives at the storage interface, a check is performed to see if its identifier is present in the Bloom filter. If the answer is *no*, the vertex can be committed to storage without checking to see if it has previously been inserted (since Bloom filters have no false negatives). This removes a significant source of overhead. In the case that the check results in a *yes*, the cache is inspected. If the vertex's identifier is found, no further action is required. This eliminates a second source of overhead. If the identifier is not found in the cache, the database will be queried to check for the presence of the identifier. The vertex is only added to the database if the identifier is not found.

Note that most Bloom filter false positives are corrected during cache checks if there is high temporal locality. A vertex may be falsely reported as present in the Bloom filter but be absent from the cache. In this case, the database check detects this, and adds the vertex to the persistent store. Consequently, the storage screening improves performance without loss of any provenance records.

5. Conclusion

SPADE is middleware for collecting, filtering, storing, and querying provenance. When used with large datasets of Bitcoin transactions and operating system activity, we encountered several challenges. The first involved the tension between collecting more detail and abstracting the provenance for specific applications. This was addressed with transformers, a new point of extensibility. In both case studies, they allowed detailed provenance to be collected while automatically pruning query responses. The second challenge derived from performing provenance integration with filters, which assume loose temporal synchronization between streams. Content-based storage and careful data modeling were combined to allow integration without any stream synchronization assumptions. The third challenge arose from the growth in size of storage subsystem memory-resident data structures. This was ameliorated with the use of a Bloom filter and cache to significantly reduce the memory required.

Acknowledgments

This work was partially funded by the U.S. National Science Foundation (NSF) under Grants IIS-1116414 and ACI-1547467 and Department of Homeland Security (DHS) Science and Technology Directorate. The views and conclusions contained herein are the author's and should not be interpreted as representing the official views of DHS, NSF, or the U.S. government.

References

[1] Divyakant Agrawal, Philip Bernstein, Elisa Bertino, Susan Davidson, Umeshwas Dayal, Michael Franklin, Johannes Gehrke, Laura Haas, Alon Halevy, Jiawei Han, Hosagrahar Jagadish, Alexandros Labrinidis, Sam Madden, Yannis Papakonstantinou, Jignesh Patel, Raghu Ramakrishnan, Kenneth Ross, Cyrus Shahabi, Dan Suciu, Shivakumar Vaithyanathan, Jennifer Widom, Challenges and opportunities with big data, *Computing Community Consortium, Computing Research Association*, 2012.

[2] Sherif Akoush, Ripduman Sohan, and Andy Hopper, HadoopProv: Towards provenance as a first class citizen in MapReduce *5th USENIX Workshop on the Theory and Practice of Provenance*, 2013.

[3] Yael Amsterdamer, Susan Davidson, Daniel Deutch, Tova Milo, Julia Stoyanovich, and Val Tannen, Putting Lipstick on Pig: Enabling database-style workflow provenance, *Very Large Database Journal*, Vol. 5(4), 2011.

[4] Controlled Access Protection Profile, `https://www.niap-ccevs.org/pp/archived/PP_OS_CA_V1.d/`

[5] Dark Net Market archives, `http://www.gwern.net/Black-market%20archives`

[6] Rip Empson, Bitcoin: How an unregulated, decentralized virtual currency just became a billion dollar market, *TechCrunch*, `http://techcrunch.com/2013/03/28/bitcoin-how-an-unregulated-decentralized-virtual-currency-just-became-a-billion-dollar-market/`, 28th March, 2013.

[7] Ashish Gehani and Minyoung Kim, Mendel: Efficiently verifying the lineage of data modified in multiple trust domains, *19th ACM International Symposium on High Performance Distributed Computing*, 2010.

[8] Ashish Gehani, Minyoung Kim, and Tanu Malik, Efficient querying of distributed provenance stores, *8th ACM Workshop on the Challenges of Large Applications in Distributed Environments*, 2010.

[9] Ashish Gehani, Dawood Tariq, Basim Baig, and Tanu Malik, Policy-based integration of provenance metadata, *12th IEEE International Symposium on Policies for Distributed Systems and Networks*, 2011.

[10] Ashish Gehani and Dawood Tariq, SPADE: Support for provenance auditing in distributed environments, *13th ACM/IFIP/USENIX International Conference on Middleware*, 2012.

[11] Ashish Gehani and Dawood Tariq, Provenance-only integration, *6th USENIX Workshop on the Theory and Practice of Provenance*, 2014.

[12] Devarshi Ghoshal and Beth Plale, Provenance from log files: a BigData problem, *1st International Workshop on Managing and Querying Provenance Data at Scale*, 2013.

[13] Boris Glavic, Big data provenance: Challenges and implications for benchmarking, *2nd Workshop on Big Data Benchmarking*, 2012.

[14] Ashvin Goel, Kenneth Po, Kamran Farhadi, Zheng Li, and Eyal de Lara, The Taser intrusion recovery system, *20th ACM Symposium on Operating Systems Principles*, 2005.

[15] Nathaniel Husted, Sharjeel Qureshi, Dawood Tariq, and Ashish Gehani, Android provenance: Diagnosing device disorders, *5th USENIX Workshop on the Theory and Practice of Provenance*, 2013.

[16] Robert Ikeda, Hyunjung Park, and Jennifer Widom, Provenance for generalized map and reduce workflows, *5th Biennial Conference on Innovative Data Systems Research*, 2011.

[17] ISO/IEC 15408-1, Information technology — Security techniques — Evaluation criteria for IT security, `http://standards.iso.org/ittf/PubliclyAvailableStandards/c050341_ISO_IEC_15408-1_2009.zip`

[18] Olga Kharif, Bitcoin economy widens as parents pay digital allowance, *Bloomberg*, `http://www.bloomberg.com/news/2014-09-25/bitcoin-economy-widens-as-parents-pay-digital-allowance.html`, 24th September, 2014.

[19] Hasnain Lakhani, Rashid Tahir, Azeem Aqil, Fareed Zaffar, Dawood Tariq, and Ashish Gehani, Optimized rollback and re-computation, *46th IEEE Hawaii International Conference on Systems Science*, IEEE Computer Society, 2013.

[20] Luc Moreau, Ben Clifford, Juliana Freire, Joe Futrelle, Yolanda Gil, Paul Groth, Natalia Kwasnikowska, Simon Miles, Paolo Missier, Jim Myers, Beth Plale, Yogesh Simmhan, Eric Stephan, and Jan Van den Bussche, The Open Provenance Model core specification (v1.1), *Future Generation Computer Systems*, 2010.

[21] Satoshi Nakamoto, Bitcoin: A peer-to-peer electronic cash system, *Cryptography Mailing List*, `https://bitcoin.org/bitcoin.pdf`, 31st October, 2008.

[22] Oracle Linux Security Guide, `https://docs.oracle.com/cd/E52668_01/E54670/html/ol7-audit-sec.html`

[23] W3C PROV, `http://www.w3.org/TR/prov-overview/`

[24] Kiran-Kumar Muniswamy-Reddy, David Holland, Uri Braun, and Margo Seltzer, Provenance-aware storage systems, *USENIX Annual Technical Conference*, 2006.

[25] RedHat Enterprise Security Guide, `https://access.redhat.com/documentation/en-US/Red_Hat_Enterprise_Linux/7/html/Security_Guide/chap-system_auditing.html`

[26] SPADE, `http://spade.csl.sri.com`

[27] SUSE Security Guide, `https://www.suse.com/documentation/sles-12/book_security/data/part_audit.html`

[28] Jianwu Wang, Daniel Crawl, Shweta Purawat, Mai Nguyen, and Ilkay Altintas, Big data provenance: Challenges, state of the art, and opportunities, *3rd IEEE International Conference on Big Data*, 2015.

[29] Chao Yang, Guangliang Yang, Ashish Gehani, Vinod Yegneswaran, Dawood Tariq, and Guofei Gu, Using provenance patterns to vet sensitive behaviors in Android apps, *11th International Conference on Security and Privacy in Communication Networks*, 2015.

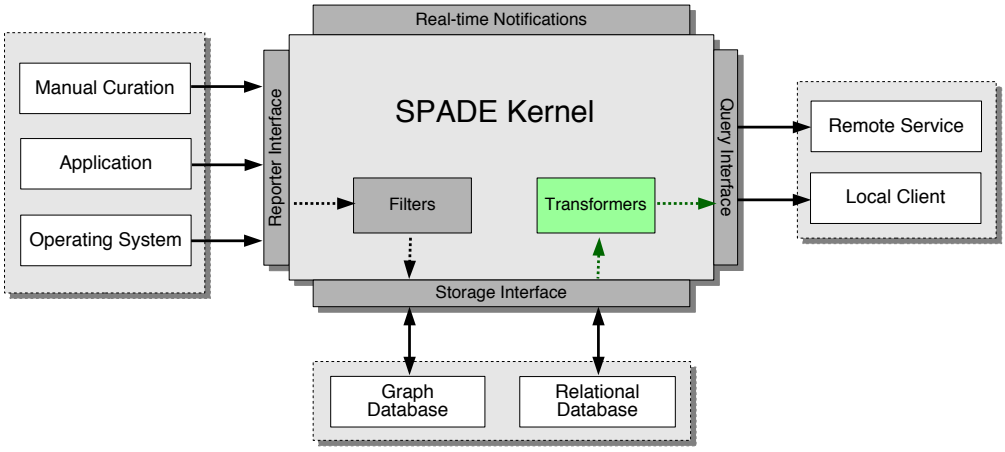

Figure 1. The SPADE kernel accepts provenance queries from local clients, interrogates the configured storage, and returns a response graph. *Transformers* operate on the response in the order they are configured. The result of the last transformer is returned to the querying client.

A. Evaluation

We report here on examples of the benefits observed after we undertook the changes in design and implementation described in Section 4.

A.1 Transformer-based Abstraction

Our primary use of transformers has been to abstract the responses to queries. We describe examples of this for the two case study domains.

A.1.1 Bitcoin

We first consider a Bitcoin query example. If a Bitcoin address (of a pseudonymous user) is found on a site soliciting donations for questionable activities, we may wish to see the network of agents whose payments have flowed to the address. If the provenance of the address is queried, it will result in a large graph of Bitcoin addresses, payments, transactions, and blocks. What is of interest is only the web of addresses.

We designed and implemented a transformer to perform the above abstraction. To understand how it works, we first describe our PROV data model for the Bitcoin blockchain. Each transaction is represented by an `Activity` vertex, that is connected via `Used` edges to input payment `Entity` vertices, and `WasGeneratedBy` edges to output payment `Entity` vertices. Each payment `Entity` is connected via a `WasAssociatedWith` edge to the `Agent` vertex of a distinct Bitcoin address. All the transactions in a Bitcoin block are connected via `WasInformedBy` edges to an `Activity` vertex that represents the block that contains them. Successive blocks are connected with `WasInformedBy` edges.

The transformer uses knowledge of this data model. It identifies all paths starting from a Bitcoin address, going to an incoming payment, then a transaction, then an outgoing payment, and finally ending at a second address. The entire path is abstracted into a single `ActedOnBehalfOf` edge from the first to second addresses. If this transformer is in place, a lineage query of an address will result in the desired response – that is, a provenance graph of addresses from which money flowed to the specified address. Table 1 illustrates the reduction in graph size from this abstraction. The untransformed graph with 4 lineage levels is shown in Figure 2. The same query with the transformer results in the graph in Figure 3.

Lineage levels	Original vertices	Original edges	Abstract vertices	Abstract edges
2	11	10	1	0
4	31	30	5	4
8	110	109	16	14
16	626	691	73	79

Table 1. This table shows the results of querying the provenance of the Bitcoin address `13Pcmh4dKJE8Aqrhq4ZZwmM1sbKFcMQEEV`, found on a site seized by the FBI. As the number of levels of lineage increases, the size of the original graph grows quickly. The transformed version excludes extraneous detail, easing analysis of the response.

A.1.2 Forensics

In our second example, we consider the challenge a system administrator faces after they detect that a web server has been compromised. In particular, they need to identify all the data that has been exfiltrated. However, if they retrieve the complete provenance of the web server process, the result provides far more detail than is necessary.

The PROV data model of the operating system reports processes as `Activity` vertices, files and sockets as `Entity` vertices, data flows to processes as `Used` edges and from

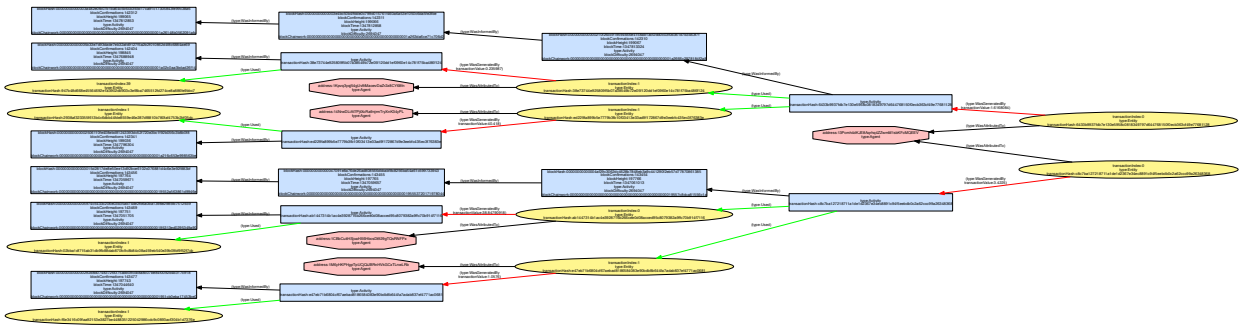

Figure 2. This graph shows the result of the lineage query with 4 levels that is described in Table 1. Note that the lineage query leverages the semantics of the provenance data model – for example, an ancestor query for an `Agent` vertex must start from its children.

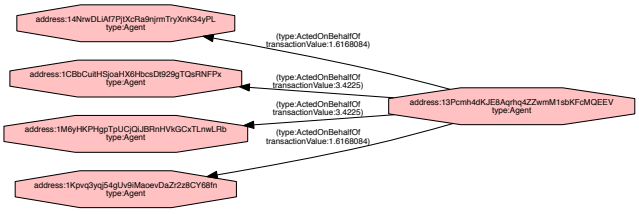

Figure 3. If the transformer is used during querying, the graph in Figure 2 is abstracted to the one shown here. As seen in Table 1, the number of abstracted vertices and edges is significantly smaller, improving comprehensibility.

them as `WasGeneratedBy` edges, and process hierarchy relationships with `WasInformedBy` edges. Depending on the application, this may provide more information than is needed.

To abstract the query response, a number of transformers can be applied. The *temporal traversal* transformer leverages knowledge of the *sink* vertex (from which the provenance query starts) and the timestamps on the edges. As it traverses ancestral dependencies, it eliminates incoming edges with timestamps later than outgoing ones. The *no versions* transformer merges different versions of the same file (that may be present due to intermediate modifications). Finally, the *merge I/O* transformer combines all reads and writes to the same file or socket into a single edge.

Transformer	Vertices	Edges
None	1969	2831
+ Temporal traversal	1061	1114
+ No versions	9	59
+ Merge I/O	9	8

Table 2. This table shows the result of querying the provenance of a file read by a web server. After the application of three transformers, the graph is small enough for a system administrator to study manually.

Table 2 shows that a query response graph with thousands of edges is reduced to one with less than a dozen. Note that none of these transformations affect the correctness of query response for the purpose of identifying the set of files that may have been leaked through the web server. This happens transparently (after the transformers have been installed in the SPADE kernel).

A.2 Temporally Desynchronized Integration

Our initial mapping of the Bitcoin blockchain to a provenance data model was not content-based integratable. This was because outgoing payments were represented along with the recipient's address in a single vertex, while vertices for incoming payments did not have an associated address (since this is not explicitly reported in a Bitcoin transaction). The reporter's memory requirement grew continuously as the ingestion proceeded, preventing the process from completing.

The mapping was changed to factor out the Bitcoin address into a separate `Agent` vertex. The residual details of outgoing payments are represented in `Entity` vertices, which now have an analogous structure to incoming payments. The new model allowed content-based integration to be utilized. In particular, this allowed outgoing payments from many years earlier to be reconciled with current incoming payments without maintaining any memory-resident state.

A.3 Screened Performance

Section 4.3 explained the three-level screening approach used prior to inserting provenance elements in persistent storage. In particular, a Bloom filter was used as a primary screen, a FIFO cache as the secondary screen, and database queries as the third level. Prior to the introduction of the Bloom filter screen, the ingestion of Bitcoin blockchain records would not complete. The system would fail after running out of memory.

We ran the ingestion framework with the Java virtual machine's heap size set to a maximum of 512 MB. Ingestion details were collected for the first 200,000 Bitcoin transac-

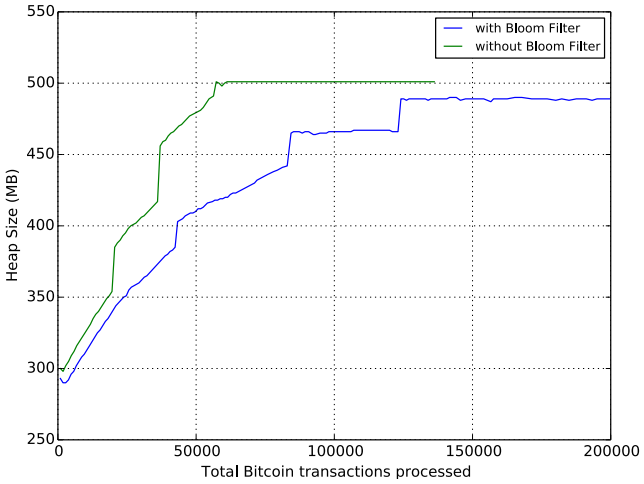

Figure 4. Without the Bloom filter screen, the memory requirements continuously grow till they cause the ingestion to fail.

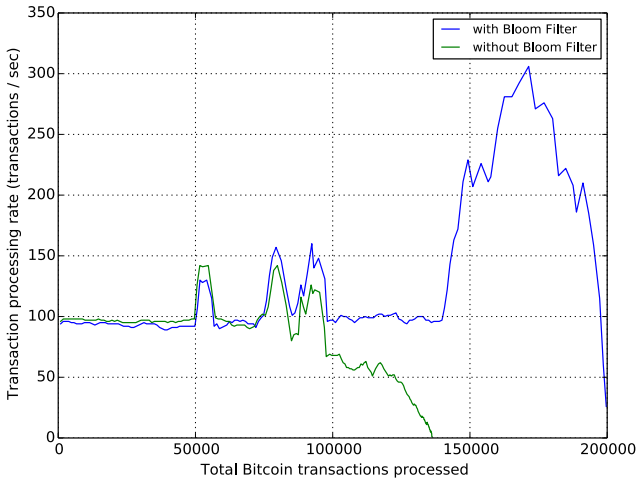

Figure 5. Without the Bloom filter screen, the transaction processing rate drops when memory pressure grows. In contrast, with the screen the rate scales to much higher loads.

tions. Figure 4 shows the amount of memory used with and without the Bloom filter screen. Without the filter, the heap rapidly grows to use all available memory, and then causes the ingestion to fail after 136,164 transactions have been processed. With the filter, the ingestion runs to completion. (Using a larger heap size simply results in the ingestion failing later.)

The number of transactions in each block affects the rate at which they can be ingested. This results in the spikes in Figure 5. Of greater interest is what occurs in the case without the use of a Bloom filter screen. As the memory pressure increases (when the heap cannot grow any larger), the rate of transaction processing drops until the ingestion process fails. In contrast, when the Bloom filter screen is deployed, the transaction processing continues and scales with the load.

B. Related Work

Data lineage for forensic analysis on individual hosts has been studied for over a decade, with Taser using it for intrusion recovery [14]. System support for tracking provenance on a single machine also dates back to a similar timeframe. A prominent example of this is PASS [24].

The first version of SPADE was developed in 2008. It used a model of decentralized provenance collection to allow it to scale to large distributed systems. Several optimizations, including the use of Bloom filters for efficient distributed queries, were developed by 2010 [7].

In late 2011, a group of prominent researchers initiated a discussion on the challenges of *big data*. This resulted in a Computing Research Association white paper [1], which identified provenance tracking as a significant issue. It was already being studied [16] in the context of MapReduce workflows that were being used to process large data sets. In 2012, Glavic coined the term *big provenance* [13] to refer to the provenance of *big data*. By 2013, the use of log files for tracking provenance was understood to be a big data problem [12].

Development of the second generation of SPADE [10] commenced in 2010. By 2012, it supported the use of operating system audit logs from Android, Linux, Mac OS X, and Windows. However, the logs were typically limited to the length of short experiments. Efforts to scale SPADE were limited to distributed query optimization, rather than analyzing the performance on individual hosts in the decentralized system.

Wang et al. [28] survey the state of the art in workflow provenance tracking, including Kepler [], RAMP [16], HadoopProv [2], and Lipstick [3]. They identify four challenges for such systems – storing the provenance records, minimizing the runtime overhead of provenance collection, integrating distributed provenance, and reproducing an execution from the workflow. We have previously reported on SPADE's runtime overhead [10], support for provenance integration [9, 11], and use for re-execution [19].