Tracking and Sketching Distributed Data Provenance

Tanu Malik
University of Chicago
Chicago, IL
Email: tanum@ci.uchicago.edu

Ligia Nistor
Carnegie Mellon University
Pittsburgh, PA
Email: lnistor@cs.cmu.edu

Ashish Gehani
SRI International
Menlo Park, CA
Email: ashish.gehani@sri.com

***Abstract*—Current provenance collection systems typically gather metadata on remote hosts and submit it to a central server. In contrast, several data-intensive scientific applications require a decentralized architecture in which each host maintains an authoritative local repository of the provenance metadata gathered on that host. The latter approach allows the system to handle the large amounts of metadata generated when auditing occurs at fine granularity, and allows users to retain control over their provenance records. The decentralized architecture, however, increases the complexity of auditing, tracking, and querying distributed provenance. We describe a system for capturing data provenance in distributed applications, and the use of *provenance sketches* to optimize subsequent data provenance queries. Experiments with data gathered from distributed workflow applications demonstrate the feasibility of a decentralized provenance management system and improvements in the efficiency of provenance queries.**

I. Introduction

The provenance of a piece of data is of utility to a wide range of applications. This is demonstrated by numerous research and industrial initiatives to develop "provenance-aware" applications [21], [17], [9], [8], [24], [15]. Of particular interest are workflow-based applications in which the data is distributed among several heterogeneous nodes in autonomous domains, and the resulting data is often combined to make useful analyses. A key issue in such systems is recording provenance metadata with minimal overhead, particularly when usage of data is tracked at a very fine granularity, and the workflows dictate the movement of data across domain boundaries. An equally important question is how to query this metadata efficiently, in real time, especially when subsequent workflow runs depend on querying the system state observed in previous workflow runs.

Current systems [21], [8], [17], [9], [1] generate provenance metadata locally, and periodically submit it to a central provenance management service. While such a service ensures the consistency of provenance metadata and ease of managing and querying the resulting metadata, it is increasingly being observed that the centralized approach introduces a large network overhead for the provenance-aware system [10]. Provenance metadata, when audited at fine granularity, grows exponentially in the number of recorded steps. Often it grows larger than the actual data [4]. Transporting large amounts of metadata to a central service introduces significant system-wide network overhead. This delays the recording and querying of provenance information, reducing system responsiveness.

To lower the overhead introduced by provenance recording systems, provenance metadata, like data, must be distributed, especially when large volumes of it are being generated. A distributed provenance model has several attractive advantages, such as (i) low-latency access to provenance metadata about local data, (ii) obviating the need for synchronization with a central service after operating while disconnected from the network, and (iii) the maintenance of user privacy — that is, computing nodes and users retain complete control over their own data provenance records. The latter aspect is particularly attractive for distributed healthcare applications and workflow-based systems in which the data is distributed among several hosts in autonomous domains.

A distributed provenance model introduces challenges as well. The first question that arises is how to track the movement of files so that there is no loss of coupling between the file content and the associated metadata. Even if a file moves and its metadata does not (due to its large size), a user should be able to subsequently retrieve the file's associated metadata. Further, gathering application-agnostic provenance requires fine-grained auditing of processes and network connections. Ideally, the auditing should not require modification of extant user programs. Another challenge is how to efficiently trace distributed provenance. The audited metadata can be viewed as a directed graph data structure. Tracing a path in a directed graph by recursively querying antecedents is known to be a computationally expensive operation [12], [3]. In the case of distributed provenance, it becomes expensive in terms of network operations as well, since part of the provenance metadata is likely to be located remotely.

We describe a decentralized provenance collection system in which each computer maintains the authoritative repository of the provenance gathered on it. The system performs fine-grained, passive monitoring of specific system calls. This is achieved with a user-space driver for the Linux kernel that intercedes on filesystem-related calls to record data flow between processes and the filesystems. On each computer, the collected provenance metadata is stored locally in a relational database. Applications are oblivious to the provenance collection and metadata distribution. To answer distributed provenance queries transparently and efficiently, the system relies on two crucial mechanisms for optimizing query execution: (i) local nodes store partial provenance data from other computers, and (ii) when the remote provenance data is not available, local nodes are assisted with *provenance sketches* to efficiently determine the other computers that need to be contacted to answer provenance path queries. The system is being developed as part of the Support for Provenance Auditing in Distributed Environments (SPADE) project [24], which aims to transparently generate, certify, and validate Grid data lineage.

We provide background information about provenance auditing systems in Section II. Section III outlines the data model of the distributed repositories we use to store the fine-grained provenance information. Section IV outlines mechanisms for improving the latency of provenance queries in wide-area systems. Section V reports our experimental results.

II. Background

Provenance auditing systems can be divided into three categories, depending upon whether (i) both data and metadata are centrally located, (ii) data is distributed but metadata is transported to a central location on a periodic basis, or (iii) both data and metadata are distributed.

The central data and metadata model was introduced by the Lineage File System (LFS) [22], which inserts *printk* statements in a Linux kernel to record process creation and destruction, operations to open, close, truncate, and link files, initial reads from and writes to files, and socket and pipe creation. The output was periodically transferred to a local SQL database. Harvard's Provenance-Aware Storage System (PASS) [21] audits a superset of the events monitored by LFS, incorporating a record of the software and hardware environments of executed processes. It provides a tighter integration between data and metadata by storing its records using an in-kernel port of Berkeley DB [14]. Both LFS and PASS are designed for use on a single node, although their designs can be extended to the file server paradigm by passing the provenance records (and queries about them) from the clients to the server in the same way that other metadata is transmitted. This architecture is also employed by the PASOA project [17]. The more recent ES3 model [9] extracts provenance information automatically from arbitrary applications by monitoring their interactions with their execution environment and logs them to a customized database. While ES3 records at a much coarser granularity than PASS, it follows the same centralized model of metadata logging.

In most workflow execution systems [23] output data and metadata are transported to a central location. A disadvantage of the workflow-based approach for collecting metadata is the requirement that computations be restricted to those that are expressible in a specific workflow language. In addition to limiting the scope of possible computations, this requires users of these systems to master and then exclusively use a specific workflow authoring environment [9].

In several systems, provenance needs to be traced across multiple system boundaries. Such a requirement has been described in centralized model systems, such as PASS [16] and ES3 [9], and several distributed healthcare [2] and e-commerce applications [11]. Distribution of provenance metadata is primarily to allow coordination of data between several heterogeneous and autonomous information systems. In Grid environments, the distribution of metadata is considered necessary for efficiently answering queries about data. For example, the Replica Location Service (RLS) [6] provides a mechanism for registering the existence of replicas and discovering them. Its metadata lookup service is distributed, reducing the update and query load, and it relies on periodic updates to keep its state from becoming stale. In another example, the Storage Resource Broker is a federated database that stores metadata as name-value pairs and is divided into zones for scalability [20].

Efficient schemes for querying provenance data have also received considerable attention. Harvard's PQL [13] describes a new language for querying provenance and leverages the query optimization principles of semi-structured databases. IBM researchers have proposed a provenance index that improves the execution of forward and backward provenance queries [15]. Query optimization techniques on compressed provenance data have also been considered [12] recently. In all these methods, the underlying architecture is of a single central provenance store. In contrast, we describe challenges and techniques for querying distributed data provenance.

III. Recording Distributed Provenance Metadata

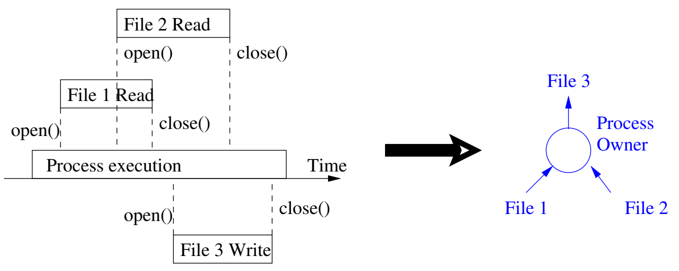


Fig. 1. By tracking the data flow between processes and the filesystem, a provenance graph representation can be automatically created.

In a distributed environment each computer has the freedom to maintain an independent filesystem and accompanying namespace, and yet data can be shared across organizational boundaries. The provenance recording infrastructure must overlay a coherent framework that facilitates reasoning about the origins of data in such a distributed environment. In particular, the infrastructure must track data flows within a host — that is, intra-host dependencies, and across hosts — that is, inter-host dependencies. We now describe how we record both intra-host and inter-host dependencies in SPADE.

Recording provenance by tracking data flows requires the system to (i) identify the sources and consumers of each piece of data, and (ii) define the granularity at which a piece of data will be tracked. On a single host, the immediate source of a piece of data will be a process, which may in turn (recursively) have used data written by other processes that have executed on the same host. In addition to the data flowing within a single host, processes may have read data from other hosts through network connections. In such an event, the provenance of any data modified by a process must also include the provenance of the data read from the remote host. We adopt the convention of identifying data by both its location in the system and the time at which it was last modified.

The granularity at which we track the provenance of a data object affects the overhead that will be introduced in the system. The advantage of finer-grain auditing, at the level of assembly instructions or system calls, for example, is that information flow can be traced more precisely, allowing an output's exact antecedents to be ascertained by reconstructing the exercised portion of the control flow graph of the relevant process. The disadvantage is that the system's performance will perceptibly degrade and the monitoring will generate large volumes of provenance metadata. Since persistent data is managed at file granularity, a reasonable compromise on the level of abstraction at which to track data provenance is to define it in terms of files read and written.

A. Intra-host Dependencies

We utilize the following elements to model intra-host dependencies in a provenance graph:

1) *File vertices* include various attributes associated with a file, such as its pathname in the host's filesystem, the size of the file, the last time it was modified, and a hash of its contents. When the provenance of a file is being discussed, the *sink* of the associated provenance graph will be the vertex corresponding to the file. We adopt the convention of identifying a file using both its logical location and its last time of modification to disambiguate different versions of the same file, which avoids cycles in the provenance graph. The SPADE schema for recording file-related provenance metadata is shown in Table I. The *Hash* field is used to store a Bloom filter containing the hashes of the fully specified names of all the files that are ancestors.
2) *Process vertices* contain a range of attributes, such as the name of the process, its operating system identifier, owner, and group. Each vertex can also include aspects such as the parent process, the host on which the process is running, the creation time of the process, the command line with which it was invoked, and the values of environment variables. We do not version process vertices though that would eliminate some false dependencies. The SPADE schema for recording process-related provenance metadata is shown in Table II.
3) *Edges* in a provenance graph are directed, signifying the direction in which data is flowing. An edge from a file vertex indicates that the file was read, while an edge into a file vertex indicates that the file was modified. Analogously, an edge leading into a process is a read operation performed by the process while an edge out of a process vertex is a write operation. Consequently, read and write operations to and from the filesystem by a process can be modeled by a data flow graph, as depicted in Figure 1.

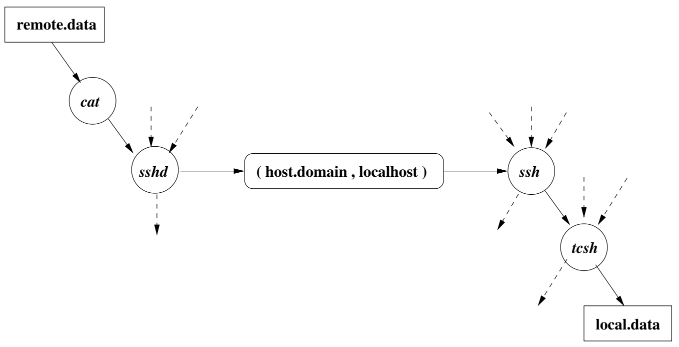


Fig. 2. Each vertex shown with a circle represents the execution of a process, while every vertex shown with a rectangle represents a file. A network vertex, depicted using a rectangle with round corners, represents a data flow through a protocol such as *ssh*, FTP, HTTP, or Java RMI.

In the context of provenance, we define the semantics of a *primitive operation* to be an output file, the process that generated it, and the set of input files it read in the course of its execution. For example, if a program reads a number of data sets from disk, computes a result and records it in a file, a primitive operation has been performed. If a process modifies a number of files, a separate instance of the representation is used for each output file.

Primitive operations are combined into a *compound operation*. For instance, if the result of appending together several data sets (by a program such as UNIX *cat*) is then sorted into a particular order (using another program, such as UNIX *sort*, that executes as a separate process), then the combination of appending and sorting is a compound operation. Thus, the provenance of every file can be represented by a compound operation that is a directed acyclic graph, consistent with the model used by Grid projects [26].

B. Inter-host Dependencies

We now consider a simple example where an operation spans multiple hosts. A user with identity **user** on the machine named **host.domain** uses *ssh* to connect to the remote host and run the UNIX *cat* program to output the contents of the file **remote.data**. The output is redirected into the file **local.data** in the filesystem of the host where the *ssh* command was invoked. This effectively copies the contents of the remote file to the local file.

```
% ssh user@host.domain cat remote.data > local.data
```

Similar commands and analogous file transfer utilities like *sftp*, FTP, or GridFTP are commonly used in large distributed

File Table

Field	Type
LFID	int(11)
Host	varchar(256)
IP	char(16)
FileName	varchar(256)
Time	datetime
NewTime	datetime
RdWt	int(11)
LPID	int(11)
Hash	varchar(256)
Signature	varchar(256)
SourceLFID	int(11)
Remote	int(11)

TABLE I
RECORDS OF FILES READ OR WRITTEN.

Process Table

Field	Type
LPID	int(11)
Host	varchar(256)
IP	char(16)
Time	datetime
PID	int(11)
PID_Name	varchar(256)
PPID	int(11)
PPID_Name	varchar(256)
UID	int(11)
UID_Name	char(32)
GID	int(11)
GID_Name	char(32)
CmdLine	varchar(256)
Environ	text

TABLE II
RECORDS OF EXECUTED PROCESSES.

Network Edge Table

Field	Type
LNID	int(11)
LocalHost	varchar(256)
LocalIP	char(16)
LocalPort	int(11)
RemoteHost	varchar(256)
RemoteIP	char(16)
RemotePort	int(11)
Time	datetime
LPID	int(11)

TABLE III
RECORDS OF NETWORK CONNECTIONS.

computations to move input data to idle processors and to retrieve the results after the execution completes. If the provenance tracking was restricted to inter-host dependencies, queries about the provenance of the file **local.data** would not be able to establish its connection to the file **remote.data** on the machine **host.domain**.

One approach to addressing the gap described above is to record information about the host on which each process runs and where each file is located. Users can then be provided a mechanism for transferring the provenance metadata when a file moves from one computer to another. Records that refer to the part of the provenance graph that originated on a remote host will be explicitly disambiguated using the *host* attribute. While this scheme ensures that all provenance queries can be answered at the destination host, it incurs considerable storage overhead [10].

An alternate approach would avoid replicating the provenance records at the destination host to which the file is being transferred. Instead, the provenance store at the destination would be provided with a pointer back to the relevant provenance metadata on the source host. However, provenance queries at the destination would require the source hosts to be contacted, slowing the response time and decreasing reliability (remote hosts may be unreachable).

In the above example, a distributed data flow takes the form of a file transfer. In practice, data may also flow through network connections directly from one process to another, as is the case in service-oriented architectures. In such systems, a series of HTTP calls is made from one host to another, each passing XML documents that include requests and arguments, and corresponding XML responses with return values. To accommodate such flows, we introduce a fourth type of element in provenance graphs — the *network vertex*. The provenance-related metadata for a TCP or UDP connection, as visible from either endpoint, can be recorded with the schema shown in Table III.

Figure 2 depicts a simplified version of the provenance graph for the file **local.data** that would arise after execution of the *ssh* command described earlier. The key point to note is that the provenance vertex for the network connection (between *ssh* and *sshd* in the example) can be independently constructed by both the hosts at the two ends of the network connection. This allows complete decentralization of the provenance recording in the distributed system, with each host's provenance infrastructure operating independently. At the same time, the provenance records generated can be pieced together to yield a coherent and complete reconstruction of the distributed data flows.

IV. QUERYING PROVENANCE

SPADE [24] allows a user to ask a wide range of questions about the files read and written by programs in a system. Several of these queries are similar to the queries described in the First Provenance Challenge of the International Provenance and Annotation Workshop [18]. Examples of SPADE queries (that can be specified with SQL) are

- *Which program was used to create this file?*
- *When the program ran what were the other files it wrote?*
- *What files did the program read?*
- *Could any data have flowed from this file to that file?*
- *What is the sequence of process executions, and files read and written that led to the flow?*

The above queries can be classified into queries that require access to (i) the entire provenance graph of the output file, (ii) just a subgraph of the provenance of a vertex, or (iii) a path in the provenance graph between specific input and output vertices.

We have developed a SPADE query tool to enable the above-described queries. The query tool interacts with the local relational provenance store using the SQL language. A relational store was chosen because of the wide adoption of relational databases in distributed environments, even though the SQL query language does not provide native support for expressing graphs and paths [13]. To address this limitation, the SPADE query tool includes a set of functions allowing users to (i) describe a file vertex and receive its entire lineage, (ii) describe a file vertex, provide a threshold k, and receive the lineage subgraph corresponding to the last k levels, and (iii) describe two file vertices and receive the lineage path between them. More complex queries, such as using regular expressions to specify paths between nodes, can be constructed using the SQL interface to the provenance store. A future version of the lineage query tool will support richer query functionality, including the specification of subgraphs and path queries parameterized with regular expressions.

Current provenance querying infrastructures [1], [4] assume access to the entire provenance graph before running a query on it. This is a safe assumption if all the provenance metadata is stored centrally. However, in a distributed environment, a typical provenance graph of a file spans numerous hosts. This necessitates a commensurate number of (high latency) network connections to reconstruct the entire provenance record. Further, if the user is interested in only a single path, then reconstructing the entire distributed graph is inefficient and degrades query performance significantly. Consider the case where the provenance graph consists of a tree with t levels and has a fan-in of f at each vertex (except the leaves). A path query needs information from at most t hosts rather than all $O(f^t)$ present in the entire graph. As f and t grow, retrieving metadata from as many as t hosts instead of as many as $O(f^t)$ computers reduces the delay substantially.

We now describe a strategy to improve the efficiency of distributed provenance query execution using *sketches* of provenance graphs, which are space-efficient representations of the lineage of a file. Each provenance sketch is propagated to the hosts of the file's descendants. A provenance query tool can use locally available provenance sketches to identify which remote hosts to contact when traversing back from sinks to sources. In particular, the sketches allow the tool to avoid following irrelevant paths back in the provenance graph. This eliminates the corresponding network connections, thereby speeding up query resolution. Section IV-A explains how to efficiently construct a provenance sketch with Bloom filters [5], and the classes of queries it supports on a single host. This is extended to the distributed case in Section IV-B.

A. Provenance Sketches on a Single Host

Consider a provenance graph on a single host, as illustrated in Figure 3. To create a sketch of the provenance graph on a single host we can use Bloom filters, which are compact data structures that provide a probabilistic representation of a set, and can handle membership queries — that is, queries that ask "Is element X in set Y?" or $inSet(Y, X)$. Given a set $S = a_1, \ldots, a_n$ of n elements, a Bloom filter allocates a vector v of m bits with all bits initially set to 0, and then chooses k independent hash functions, $h_1, \ldots, h_k$, each with range $1, \ldots, m$. To insert an element a, the bits at positions $h_1(a), \ldots, h_k(a)$ in v are set to 1. A particular bit may be set to 1 multiple times. Given a query for b, the bits at positions

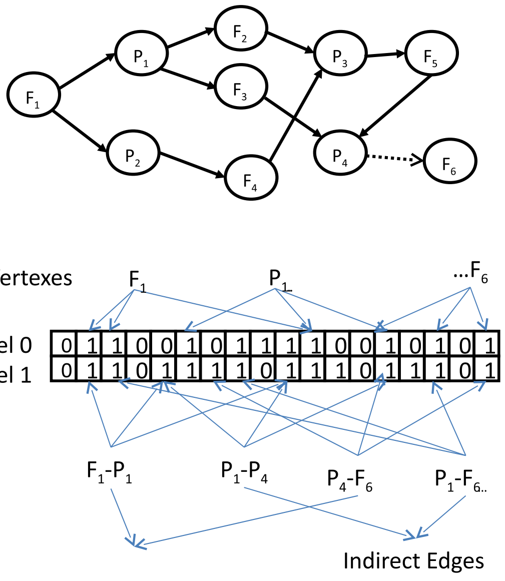


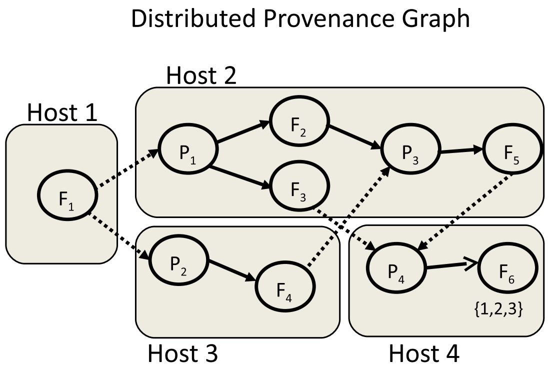
Fig. 3. Bloom filter-based *provenance sketch*.

$h_1(b), \ldots, h_k(b)$ are inspected. If any of them is 0, then b is certainly not in the set S. Otherwise, we conjecture that b is in the set. The probability that this is a false positive is $(1 - e^{-\frac{kn}{m}})^k$.

To create a provenance sketch, we can use the vertices of the graph as elements of the set S. However, this enables only vertex membership queries, such as $inSet(S, F_1)$ and $inSet(S, P_1)$, but not provenance path queries, such as $P_1/F_2/\star/P_2/F_4$, which determines if file F_2 generated by process P_1 is in lineage of file F_4, generated by process P_2. To enable provenance path queries, one alternative is to store just the edges of the graph as the set members. This does enable some path queries in which all vertices in the path are entirely specified (but not regular expression path queries). Further, if only the edges are stored, it considerably increases the rate of false positives, as seen in Section V. For instance, for the path query $P_1/F_2/\star/P_2/F_4$, a filter with edges as set members returns "true", since edges P_1–F_2 and P_2–F_4 are in the filter. However, file F_2 is not in the lineage of file F_4.

To enable all kinds of path queries, a 2-level Bloom filter construction is shown in Figure 3. The first level of the filter stores all vertices of the graph and enables set membership queries for individual vertices. The second level summarizes the *direct* and *indirect* edges of the graph as members of the Bloom filter. Direct edges are those edges that are specified in the graph, and indirect edges are obtained by computing the transitive closure [7] over the direct edges of the provenance graph. Such a construction does not introduce any false positive probability beyond what is inherent to the filter.

Our provenance sketch can be updated efficiently when a new vertex is added to the graph. The first level of the filter is updated by adding the unique specification of the vertex to it. The second level is updated with direct edges obtained by querying the local database for the parents of



Global Network Edge table

Host	LFID	LPID	Host
Host 1	F_1	P_1	Host 2
Host 1	F_1	P_2	Host 3
Host 3	F_4	P_3	Host 2
Host 2	F_3	P_4	Host 4
Host 2	F_5	P_4	Host 4

Provenance Sketch Edges

Host	Start	End	Check
Host 4	F_6	P_4	Yes
Host 2	P_1	F_3	Yes
Host 2	P_1	F_5	Yes
Host 2	P_3	F_3	No
Host 2	P_3	F_5	Yes
Host 3	P_2	F_4	Yes

Fig. 4. *Provenance sketches* facilitate path query responses without reconstructing the entire provenance graph.

the new vertex. To obtain indirect edges, we calculate the incremental transitive closure by checking set membership queries between the parents of the new vertex and the other vertices of the graph. If any of the set membership queries are true, then by transitivity we add indirect edges to the filter between the new vertex and the other queried vertices. (Periodically, the filter is reconstructed with indirect edges that are obtained by performing the computing-intensive transitive closure procedure on the provenance graph itself. Since the filter is not used to decide whether to add indirect edges, the false positive rate is reduced.)

B. Provenance Sketches on Multiple Hosts

A provenance sketch enables path queries that span a single host. For path queries that span multiple hosts, a provenance sketch is insufficient as it does not include network vertices. To determine if a path exists between two vertices, each on a different host, it is important for a host to know three kinds of information: (i) the hosts of the starting and ending vertices (specified in the path query), (ii) all the edges that cross hosts, and (iii) direct and indirect paths within each host. The set of cross-host edges can be obtained by combining the network connection entries, as would be stored in Table III, for all the hosts on which any ancestor vertices originate. Each host summarizes its direct and indirect edges in a provenance sketch and makes the sketch available to other hosts. We now demonstrate through an example how a query tool can determine, through locally available information, if a path exists between vertices on two different hosts and which hosts to contact to obtain the path.

Consider the provenance graph of Figure 3, now distributed on four hosts in Figure 4. Assume that the user specified a regular expression path query "$F_1/\star/F_6$" on `Host 1`. To determine if a path exists, SPADE must first determine if the start and end vertices specified in the query are local to the host. If the specified start or end vertex is not local, the query is forwarded to the relevant host (using the provenance sketch), which is `Host 4` in our example. The host then queries the local database with the start or end vertex, which is F_6 in our example. It obtains the set of hosts on which all ancestor vertices originated. (The list is shown in parentheses in Figure 4.)

A self-join is recursively performed on the table of network connections, described in Section III-B, to find a set of cross-host edges that span the path from the start vertex to the end vertex of the query. The internal edges on a host between two cross-host edges are resolved using the corresponding host-specific provenance sketch. All the subpaths found are connected together to provide a complete path from the start to end vertex.

V. Experiments

Our experiments are conducted on provenance metadata that was gathered by using SPADE to monitor the workflow of a large distributed application in SRI International's Speech Technology and Research Laboratory [25]. The application workload originated as part of the NIGHTINGALE project [19], which allows monolingual users to query information from newscasts and documents in multiple languages. The objective of the project is to produce an accurate translation. NIGHTINGALE aims to achieve this with a workflow that specifies the tools that will transform the inputs using automatic speech recognition algorithms, machine translation between languages, and distillation to extract responses to queries. However, there is no canonical algorithm for each of these steps, necessitating a choice between the tools used. The speech scientists use accompanying metadata to estimate which combination of available tools will produce the most accurate result. This is further complicated by the fact that the tools have multiple versions and are developed in parallel by experts from 15 universities and corporations. Finally, the choice of which specific version of a tool to use depends on the outcome of previous workflow runs.

A representative application workload executed for two hours with SPADE collecting provenance metadata about the processes that ran and files that were accessed. The resulting provenance graph has 8621 file vertices, 9433 process vertices, a maximum fan-in of 22, maximum fan-out of 260, a depth of 17 levels, and a width of 2746 vertices.

Since the workflow was obtained from a single site (at SRI), we used domain knowledge to divide it to correspond to a distributed execution over 10 hosts with matching network connection entries. Based on the use cases of the speech

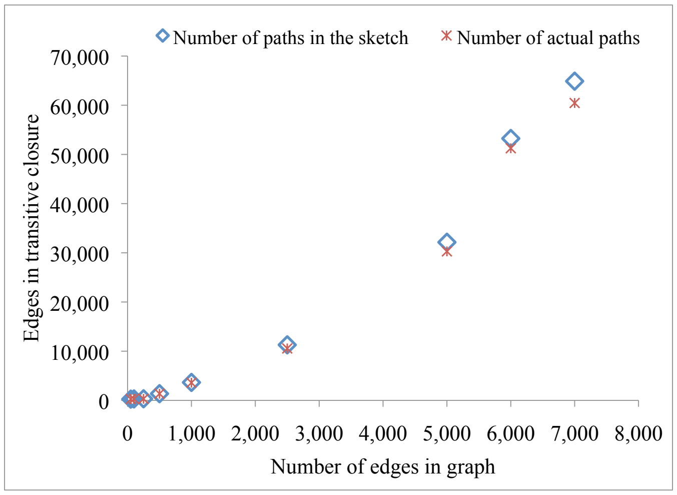


Fig. 5. Benefit of approximate incremental transitive closure.

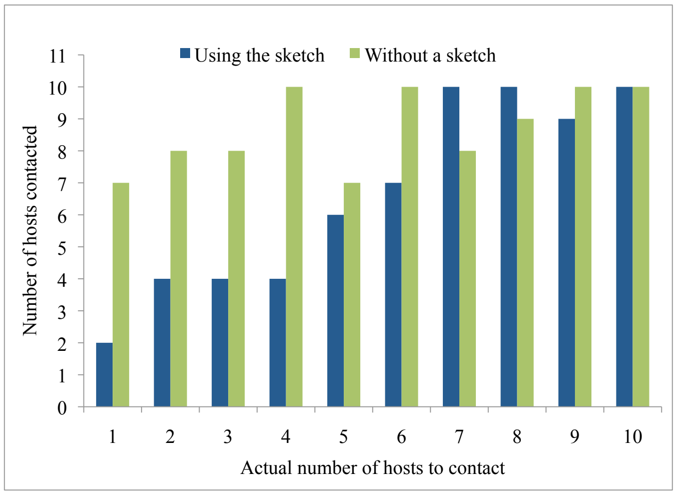


Fig. 6. Estimated number of hosts to contact using a provenance sketch.

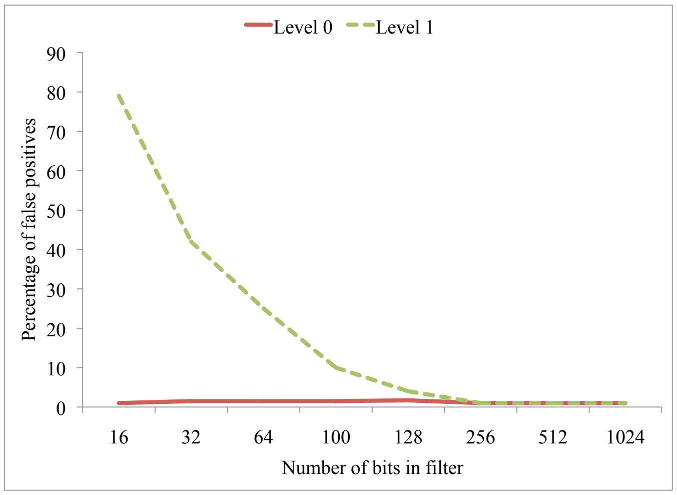


Fig. 7. Influence of filter size on the performance of the provenance sketch.

scientists, we generated a provenance query workload with three kinds of queries: (i) requests for the entire lineage of a process or file, (ii) requests for the last k levels in the lineage of a file or process, where k is chosen randomly between 10 and 20, and (iii) path queries where the path is completely specified or is described as a regular expression, with the length limited to 2. Without loss of generality, regular expression queries have start and end vertices specified. The vertices chosen follow a a Zipf distribution [27].

We consider a multihost system where each host stores a set of provenance graphs. Queries may originate at any host in the system. A client requests lineage information in the form of any of the above-described queries. As described in Section IV, the provenance sketch associated with each file contains two Bloom filters — one represents the set of all the vertices and the second summarizes the set of all the edges of the provenance of the file. The first filter facilitates routing queries only to hosts that contain relevant information. The second filter is used to determine which portion of the query can be locally matched.

We now describe experiments that test the efficiency of distributed querying using provenance sketches. The provenance sketches use Bloom filters of 100 bits with four hash functions that have an output range from 0 to 10, unless otherwise specified.

A. *Influence of Incremental Transitive Closure*

Each provenance sketch is computed incrementally and approximately — that is, whenever a new vertex (and its corresponding edge) is inserted, the current sketch is used to check if there are paths from the vertex's parent to other vertices. An indirect edge from the new vertex is added to the sketch for each path found, with the last vertex of the path used as the other end of the edge. This approach updates the transitive closure in $O(VE)$ time, which is typically much less than the $O(V^3)$ time needed to compute the transitive closure of the entire graph.

We measure the benefit of computing an incremental transitive closure instead of the complete transitive closure of the entire graph. This is done by counting the number of path queries that can be answered with the elements that are in the provenance sketch versus the number of paths that actually exist in the graph. This is done for graphs with an increasing number of direct edges but the number of vertices fixed at 10,000. Figure 5 shows the performance of the provenance sketch. Provenance sketches are more accurate over sparse graphs as the size of the closure is much smaller, reducing the false positives from the filter. However, for dense graphs the decrease in performance is sublinear — that is, doubling the number of edges in the graph decreases the sketch's accuracy by less than 5%.

B. *Reduction in Network Latency*

The dominant cost of answering distributed provenance queries comes from the network connections. To measure the reduction in network latency, we undertook the following experiment. We divided the provenance graph to correspond to its execution on 10 hosts. The x axis of Figure 6 shows the actual number of hosts that need to be contacted by a given set of queries. The actual number of hosts contacted for each set of queries is shown along the y axis. We also plot the maximum number of hosts that would have to be contacted if no provenance sketch was available locally.

Figure 6 shows that using provenance sketches reduces the number of hosts that must be contacted to answer a query. The difference in the actual number of hosts that a query contacts and the number of hosts that a sketch estimates is not significant, except for a few cases. (For example, when the actual number of hosts that needs to be contacted is 3 and 5, the sketch reports 5 and 7. This is an artifact of our distribution of the provenance graph onto multiple hosts, where most queries have vertices from 3 or 5 hosts along their paths, and queries are of length at most 2.)

C. Influence of Filter Size

Finally, we examined the influence of the size of the Bloom filter on the false positive probability of each level of the sketch. We executed the query workload with different-size filters, ranging from 16 to 1000 bits. Figure 7 shows the resulting false positive rate.

VI. Conclusion

SPADE is a system for auditing fine-grained provenance in distributed environments. We described how SPADE stitches together provenance graphs collected at different hosts. Bloom filter-based summary data structures are used to optimize the execution of distributed provenance queries. SPADE was used to capture the provenance metadata of a distributed application workload that originated in the NIGHTINGALE project [19]. Using the resulting records, we analyzed and reported on the efficiency of using Bloom filter-based sketches for distributed provenance queries, including those with regular expressions.

Acknowledgement

We thank Basim Baig and Dawood Tariq for help with provenance graph analysis.

This material is based upon work supported by the National Science Foundation under Grant OCI-0722068. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation.

References

[1] P. Agrawal, O. Benjelloun, A.D. Sarma, C. Hayworth, S. Nabar, T. Sugihara, and J. Widom, Trio: A system for data, uncertainty, and lineage, 32nd International Conference on Very Large Data Bases, 2006.

[2] S. Alvarez, J. Vazquez-Salceda, T. Kifor, L.Z. Varga, and S. Willmott, Applying provenance in distributed organ transplant management, Lecture Notes in Computer Science, Springer-Verlag, Vol. 4145, 2006.

[3] Manish Anand, Shawn Bowers, and Bertram Ludaescher, Techniques for Efficiently Querying Scientific Workflow Provenance Graphs, 13th International Conference on Extending Database Technology, 2010.

[4] U. Braun, S. Garfinkel, D. A. Holland, K. Muniswamy-Reddy, and M. Seltzer, Issues in automatic provenance collection, Lecture Notes in Computer Science, Springer-Verlag, Vol. 4145, 2006.

[5] A. Broder and M. Mitzenmacher, Network applications of Bloom filters: A survey, Internet Mathematics, Vol. 1(4), 2004.

[6] A.L. Chervenak, N. Palavalli, S. Bharathi, C. Kesselman, and R. Schwartzkopf, Performance and scalability of a replica location service, 13th ACM International Symposium on High Performance Distributed Computing, 2004.

[7] G. Dong, L. Libkin, J. Su and L. Wong, Maintaining transitive closure of graphs in SQL, International Journal of Information Technology, Vol. 5,1999.

[8] I. T. Foster, J. S. Vockler, M. Wilde, and Y. Zhao, Chimera: A virtual data system for representing, querying, and automating data derivation, 14th International Conference on Scientific and Statistical Database Management, 2002.

[9] J. Frew, D. Metzger, and P. Slaughter, Automatic capture and reconstruction of computational provenance, Concurrency and Computation, Vol. 20(5), 2008.

[10] Ashish Gehani and Ulf Lindqvist, Bonsai: Balanced lineage authentication, 23rd Annual Computer Security Applications Conference, IEEE Computer Society, 2007.

[11] P. Groth, S. Miles, and L. Moreau, Preserv: Provenance recording for services, UK OST e-Science Second All Hands Meeting, 2005.

[12] T. Heinis and G. Alonso, Efficient lineage tracking for scientific workflows, ACM SIGMOD International Conference on Management of Data, 2008.

[13] D. A. Holland, U. Braun, D. Maclean, K. Muniswamy-Reddy, and M. Seltzer, Choosing a data model and query language for provenance, 2nd International Provenance and Annotation Workshop, 2008.

[14] Aditya Kashyap, File System Extensibility and Reliability Using an In-kernel Database, Master's Thesis, State University of New York, Stony Brook, 2004.

[15] A. Kementsietsidis and M. Wang, On the efficiency of provenance queries, 25th International Conference on Data Engineering, 2009.

[16] J. Ledlie, C. Ng, D.A. Holland, K. Muniswamy-Reddy, U. Braun, and M. Seltzer, Provenance-aware sensor data storage, 1st IEEE International Workshop on Networking Meets Database, 2005.

[17] Simon Miles, Ewa Deelman, Paul Groth, Karan Vahi, Gaurang Mehta, and Luc Moreau, Connecting scientific data to scientific experiments with provenance, 3rd IEEE International Conference on e-Science and Grid Computing, 2007.

[18] L. Moreau, B. Ludaescher, I. Altintas, R. Barga, S. Bowers, S. Callahan, G. Chin Jr, B. Clifford, S. Cohen, S. Cohen-Boulakia, S. Davidson, E. Deelman, L. Digiampietri, I. Foster, J. Freire, J. Frew, J. Futrelle, T. Gibson, Y. Gil, C. Goble, J. Golbeck, P. Groth, D. Holland, S. Jiang, J. Kim, D. Koop, A. Krenek, T. McPhillips, G. Mehta, S. Miles, D. Metzger, S. Munroe, J. Myers, B. Plale, N. Podhorszki, V. Ratnakar, E. Santos, C. Scheidegger, K. Schuchardt, M. Seltzer, and Y. Simmhan, The First Provenance Challenge, Concurrency and Computation: Practice and Experience, Vol. 20(5), 2007.

[19] Novel Information Gathering and Harvesting Techniques for Intelligence in Global Autonomous Language Exploitation, http://www.speech.sri.com/projects/GALE/

[20] A. Rajasekar, M. Wan, R. Moore, W. Schroeder, G. Kremenek, A. Jagatheesan, C. Cowart, B. Zhu, S.Y. Chen, and R. Olschanowsky, Storage resource broker-managing distributed data in a Grid, Computer Society of India Journal, Vol. 33(4), 2003.

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

[22] Can Sar and Pei Cao, Lineage File System, http://crypto.stanford.edu/~cao/lineage.html

[23] Y. L. Simmhan, B. Plale, and D. Gannon, A survey of data provenance in e-science, ACM SIGMOD Record, Vol. 34(3), 2005.

[24] Support for Provenance Auditing in Distributed Environments, http://spade.csl.sri.com/

[25] Speech Technology and Research Laboratory, SRI International, http://www.speech.sri.com

[26] Douglas Thain, Todd Tannenbaum, and Miron Livny, Condor and the Grid, Grid Computing: Making The Global Infrastructure a Reality, John Wiley, 2003.

[27] Zipf's Law, http://mathworld.wolfram.com/ZipfsLaw.html