Efficient Querying of Distributed Provenance Stores

Ashish Gehani Minyoung Kim
SRI International*
Menlo Park, CA
{ashish.gehani,minyoung.kim}@sri.com

Tanu Malik
Purdue University
West Lafayette, IN
tmalik@purdue.edu

ABSTRACT

Current projects that automate the collection of provenance information use a centralized architecture for managing the resulting metadata - that is, provenance is gathered at remote hosts and submitted to a central provenance management service. In contrast, we are developing a completely decentralized system with each computer maintaining the authoritative repository of the provenance gathered on it. Our model has several advantages, such as scaling to large amounts of metadata generation, providing low-latency access to provenance metadata about local data, avoiding the need for synchronization with a central service after operating while disconnected from the network, and letting users retain control over their data provenance records. We describe the SPADE project's support for tracking data provenance in distributed environments, including how queries can be optimized with *provenance sketches*, pre-caching, and caching.

Categories and Subject Descriptors

H.3.3 [**Information Systems**]: Information Search and Retrieval

Keywords

provenance, lineage, querying, metadata, distributed storage

1. INTRODUCTION

The provenance of a piece of data is of utility to a wide range of domains. This is demonstrated by the numerous research and industrial initiatives to develop "provenance-aware" systems [41, 34, 16, 15, 44, 27]. Of particular interest are applications in which the data and workflow are distributed among several heterogeneous and autonomous information systems and are often combined to make useful analysis. A key issue when automating the collection of provenance information is how to record the metadata, especially when it concerns data that is flowing across system boundaries. An equally important question is how to query this metadata efficiently [9, 35, 18]. Current provenance recording systems [41, 15, 34, 16, 1] use a centralized architecture for managing the resulting metadata - that is, provenance is gathered at remote hosts in a distributed system and periodically submitted to a central provenance management service that ensures the consistency of data. However, it is increasingly being observed that the centralized approach is not scalable. Provenance metadata, when audited at finer granularity, grows exponentially in the number of recorded steps. Often it becomes larger than the actual data [6]. Transporting large amounts of metadata to a central service introduces significant system-wide network overhead and creates substantial latency in responses to provenance queries [17].

Provenance metadata, similarly to data, must be distributed when large volumes of it are being handled. A distributed provenance model has several attractive advantages, such as low-latency access to provenance metadata about local data, no need for synchronization with a central service after operating while disconnected from the network, and maintenance of user privacy - that is, users retain complete control over their data provenance records. The latter aspect is particularly vital for distributed healthcare applications.

A distributed provenance model introduces challenges as well. The first question that arises is how to audit 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 (due to its large size) does not, a user should be able to subsequently retrieve the file's associated metadata. Further, tracking distributed 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 back distributed provenance. The audit 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 [20]. In the case of distributed provenance, it becomes expensive in terms of network operations as well, since the provenance metadata is unlikely to be located locally.

SPADE (Support for Provenance Auditing in Distributed Environments) [44] is a decentralized system in which each computer node maintains the authoritative repository of the provenance gathered on it. The system performs fine-grained passive monitoring with a user-space driver for the Linux kernel by interceding on specific system calls to record data flow between processes, filesystems, and network connections. For each computing node, the collected provenance data is stored locally in a relational database. Applications are oblivious to SPADE's provenance collection and metadata distribution. Distributed provenance queries are answered

*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.

HPDC'10, June 20–25, 2010, Chicago, Illinois, USA.

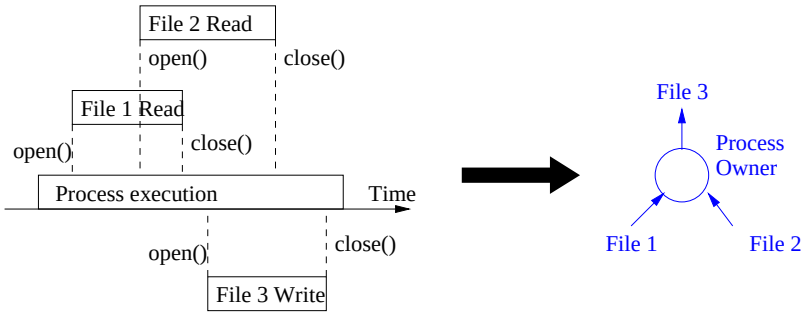

Figure 1: By tracking the data flow between processes and the filesystem, a provenance graph representation can automatically be created.

transparently and efficiently. SPADE relies on two crucial mechanisms for optimizing query execution: (i) local caches that can store provenance data from other computing nodes, and (ii) *sketches* for efficient storage of provenance metadata to answer path queries.

We provide background on provenance systems in Section 2. Section 3 outlines our data model for capturing fine-grained provenance information in distributed repositories. Section 4 outlines mechanisms for improving the latency of provenance queries in wide-area systems. We describe related caching work in Section 5 and conclude in Section 6.

2. BACKGROUND

Provenance gathering 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) [42], 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) [41] 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 [26]. 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 [34]. The more recent ES3 model [16] 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 [43] 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 [16].

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 [30] and ES3 [16], as well as several distributed healthcare [3] and e-commerce applications [19]. 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) [10] 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 [40].

Efficient schemes for querying provenance data have also received considerable attention recently. Harvard's PQL [21] 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 [27]. Query optimization techniques on compressed provenance data have also been considered [20] recently. In all these methods, the underlying architecture is of a single central provenance store. We describe challenges in querying distributed provenance where network costs dominate.

3. RECORDING PROVENANCE METADATA IN DISTRIBUTED ENVIRONMENTS

We do not assume that hosts within a distributed environment have a common filesystem, allowing each host the freedom to maintain an independent filesystem and accompanying namespace. Such distributed environments correspond to a general framework for distributed computation where data is shared across organizational boundaries. The provenance recording infrastructure then overlays a coherent framework that facilitates reasoning about the origins of data in the distributed environment. In particular, the infrastructure tracks data flows within a host - that is, intra-host dependencies, and across hosts - that is, inter-host dependencies.

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,

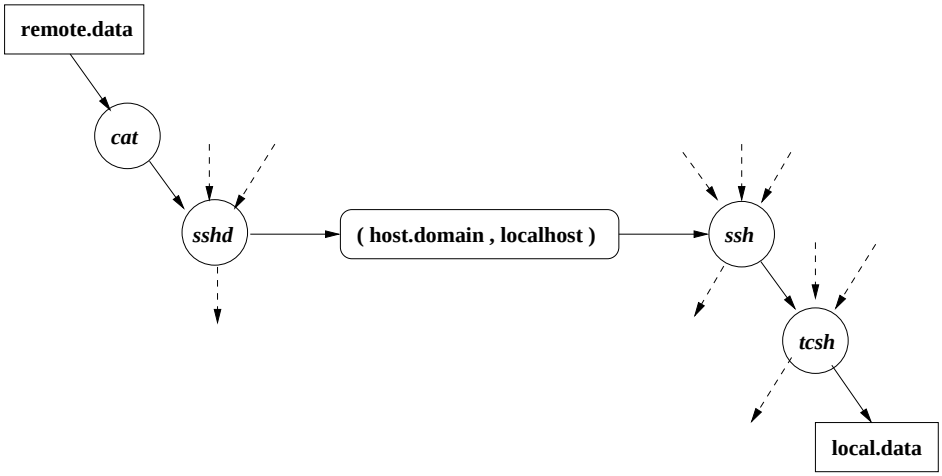

Figure 2: Each vertex shown with a circle represents the execution of a process while every vertex shown with a rectangle represents a file. A network vertex, depicted using a rectangle with round corners, represents a data flow through a protocol such as *ssh*, FTP, HTTP, or Java RMI.

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. This ensures that the previous statement holds true even if multiple processes modify a file since they will do so at different points in time if a single kernel is mediating access to the local filesystem even if there are multiple cores on the same host.

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.

3.1 Intra-host Dependencies

The first type of element in a provenance graph is the *file vertex*, which can include in it 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 *root* 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 second type of element in a provenance graph is the *process vertex*. It can contain a range of attributes, such as the name of the process, its operating system identifier, owner, and group. The vertex can also include aspects such as the parent process, host on which the process is running, creation time of the process, command line with which it was initiated, and values of environment variables.

Edges in a provenance graph are directed, signifying the direction in which data is flowing. An edge from a file vertex indicates the file being read, while an edge into a file vertex is a write to the file. 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.

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 [46].

3.2 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 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

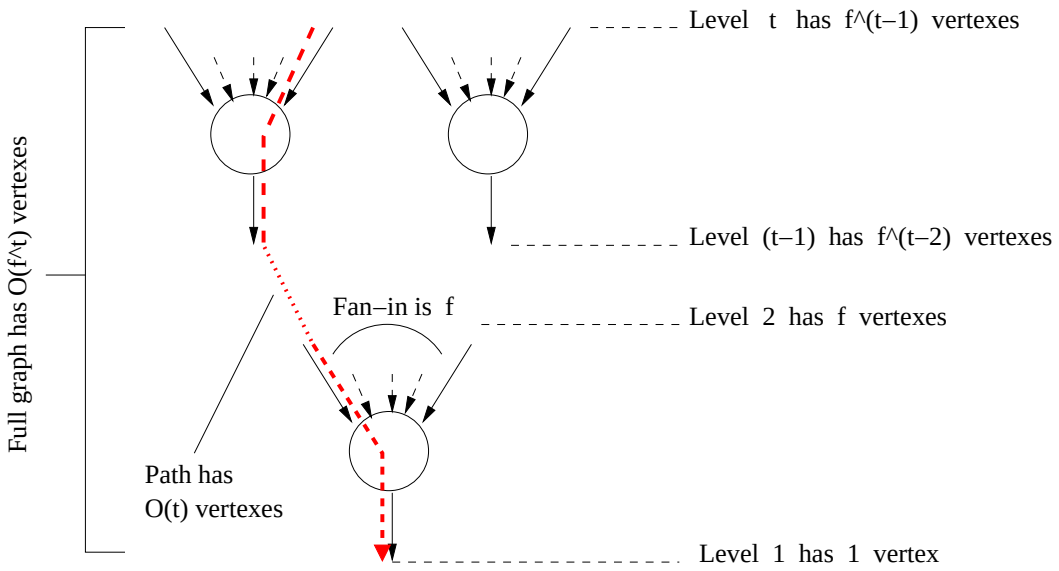

Figure 3: *Sketches* facilitate path query responses without reconstructing the entire provenance graph.

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 [17].

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 (since remote hosts may be unreachable).

In the above example, distributed data flows 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*.

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. Yet at the same time, the provenance records generated can be pieced together to yield a coherent and complete reconstruction of the distributed data flows.

4. QUERYING PROVENANCE

SPADE [44] allows a user to ask a wide range of questions about the data stored and the programs run in a system. These include queries in the Provenance Challenge of the International Provenance and Annotation Workshop [36], as well as queries such as

- 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, or (iii) a path in the provenance graph between specific input and output vertices. A user who wishes to ask such queries can employ the SPADE query tool to interact with the local provenance store. If the query cannot be resolved with local information, then the user must contact the source computers through the network. A typical provenance graph associated with a file will span numerous computers, necessitating a commensurate number of (high latency) network connections to reconstruct the entire provenance record. To improve the efficiency of executing distributed provenance queries, SPADE currently implements two approaches that reduce the latency of the queries. First, it optimizes the query execution through the use of summary structures, as described in Section 4.1. Second, it employs pre-caching of provenance records from remote hosts when files are transferred using an *overloaded namespace* [18]. Sections 4.2 and 4.3 describe pre-caching and caching strategies that are still at the design stage.

4.1 Provenance Sketches

Current provenance querying infrastructures [1, 6] assume access to the entire provenance graph before running a query on it. Since this assumption has an associated metadata retrieval cost in decentralized systems, SPADE is exploring a scheme that exploits the semantics of the query type to avoid the retrieval of provenance information when possible. The benefit of this is easily seen

Inputs:
Vertices: Provenance records in local and remote stores
Choices: Do nothing | Replicate at remote node | Transfer to remote node | Evict at both nodes
Output:
Mapping: $\forall vertex \in Vertices, \exists choice \in Choices : vertex \rightarrow choice$
Procedure:
Initialize *bestMapping* = *currentMapping*;
while (gain improves)
Unlock all vertices;
while ($\exists$ unlocked *vertex*)
Find best $vertex$ and $choice$;
Perform $choice$ and lock $vertex$;
if(gain(*currentMapping*) $>$ gain(*bestMapping*))
bestMapping = *currentMapping*;
endif;
endwhile;
currentMapping = *bestMapping*;
endwhile;

Figure 4: Strategy for pair-wise exchange of records between distributed provenance stores in the system.

through an example, depicted in Figure 3. Consider the case where the provenance graph consists of a tree with t levels and fan-in f at each vertex (except the leaves). A path query needs information from at most t computers rather than all $O(f^t)$ present in the entire graph. As f and t grow, retrieving metadata from as many as t computers instead of as many as $O(f^t)$ computers reduces the delay substantially.

One strategy to effect path query optimization is to implement *provenance sketches* that contain a space-efficient representation of the metadata attributes of a file's ancestors. When a process modifies a file, a combination of its inputs' sketches will be stored as the output's provenance sketch. Consequently, when trying to determine from which computer a particular input originated, the provenance sketches can be used to pick the correct one. This in turn allows provenance queries to be resolved by traversing back from the sink to a source without inspecting any irrelevant paths that may require network connections to computers from where the data did not originate. A salient feature of the approach is that the provenance sketches are opaque, allowing them to be propagated without compromising the privacy of the provenance sources.

4.2 Decentralized Pre-Caching

Global knowledge about the provenance of all the files in the distributed system facilitates optimal selection of where to pre-cache replicas, enabling the development of the clustering approach that we have previously described [18]. However, in many settings such access to all the metadata is not feasible. We were made aware of this fact when we began to investigate the needs of the NIGHTINGALE project [38], which aims to let monolingual users query information from newscasts and documents in multiple languages. Input data is transformed multiple times for automatic speech recognition, machine translation between languages, and distillation to extract responses to a query.

The NIGHTINGALE pipeline of operations has several steps, and they can be performed by multiple versions of software being developed in parallel by experts from 15 universities and corporations. Since the functionality of different revisions of the same tool can also differ, the description of the tool that produced a piece of data serves as an input for subsequent tools in the pipeline. This metadata is currently maintained in a file that accompanies the data. If low latency access to the provenance of data were available, maintenance of the accompanying file would be obviated. The low latency is of significance because the metadata would enable querying to determine which combinations of tools in the pipeline have yielded a better-quality output.

Since the project is a loose collaboration between researchers from independent domains, an approach that relied on access to everyone's provenance metadata *a priori* would face administrative challenges. Instead, a more scalable approach would bootstrap the system by optimizing the metadata caching within clusters where sharing was occurring in practice. Subsequently, as data crossed domains, the system could augment its record of provenance metadata using information from other domains. This would also allay privacy concerns by limiting provenance stores to communication only with domains that are trusted.

Provenance Gossip

Instead of collecting all the provenance records from the entire system and attempting to decide where replicas should be placed, the decentralized approach operates as follows. A gossip protocol can be used where provenance stores on pairs of hosts in the system periodically interact with each other. The algorithm builds on the Kernighan-Lin (KL) [28] and Fiduccia-Mattheyses (KL-FM) [14] algorithms used for circuit design. The KL-FM algorithm attempts to minimize the wires between chips when a circuit is partitioned among them. In contrast to KL-FM, our algorithm replicates a vertex in a different partition, potentially evicting an extant one, if the gain warrants the action.

Caching Choices

In the original KL-FM algorithm, the only action possible when considering two vertices from different partitions was to either swap them or do nothing. In our framework we wish to allow these but also introduce the possibility of replicating each vertex in the other partition. Further, to address the fact that there is a finite amount of storage that can be used for caching copies of provenance metadata about operations that occurred on remote hosts, we must also allow a vertex to be deleted - that is, evicted from the local cache - if it does not correspond to an operation that was performed on the local host. (Vertices of local operations must never be deleted since

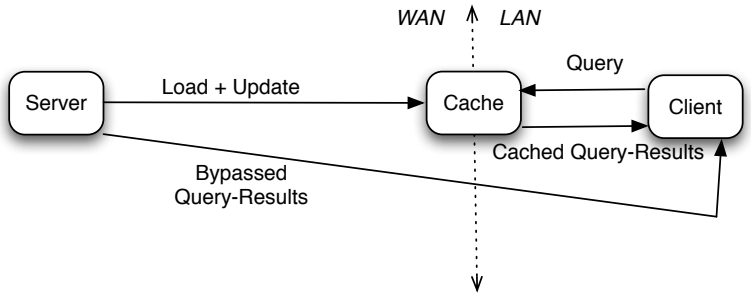

Figure 5: Provenance metadata flows between a query client and a provenance store through a cache. The cache is transparent to the client. We aim to minimize the total network traffic generated from the provenance store - that is, the sum of traffic to load and update the cache, and the query results that bypass the cache and are sent directly to the client.

they are the definitive copy that may be retrieved if all other copies have been evicted.)

Gain Function

To decide which action to take for each vertex, a *gain* function is utilized. The gain is defined as the difference between the *benefit* and *cost* functions. The KL-FM framework is agnostic to the specific functions used, allowing us to consider multiple criteria, such as query response times, host reliability, and the joint probabilities of requests originating from multiple hosts. A simple benefit function would compute the maximum likelihood that a query would use the vertex in consideration while the host from which the vertex originated is not reachable within a predefined time. Similarly, the cost could be defined as the sum of the storage used for all the vertices, weighted by their probability of being used to answer a query. The system can even be architected to use host-specific gain functions that depend on the expected application and query workloads, although the efficacy of such a strategy remains to be validated.

Mapping Algorithm

As in the original KL-FM framework, we can use two nested loops to perform the mapping of provenance vertices to associated caching and eviction actions. In the inner loop, all the actions on all the vertices are compared to estimate which would result in the greatest gain. If none would yield a gain, the one that would impose the least penalty is selected. This ensures that the algorithm is not subject to local maxima. The selected vertex is locked, signifying that it is not considered during successive iterations of the inner loop. While unlocked vertices are available, the inner loop proceeds. When it completes, the choices for each vertex are used to compare the current mapping of vertices to the computed mapping. If the new mapping yields a higher cumulative gain, it replaces the choices in the current mapping. The outer loop continues as long as gains can be improved by running the inner loop. At the outset of each iteration of the outer loop, the vertices are all unlocked. The essence of the process is illustrated in Figure 4.

4.3 Caching Between Client and Store

Clients making provenance queries may be on hosts different from the ones on which provenance is being gathered. For example, provenance may be gathered on all the nodes in a Grid cluster where a computation has been farmed out, but the desktop machines where scientists are performing data analysis may not be running a provenance auditing system. In such situations, *provenance-aware caches* can be placed between the clients and provenance stores. These provide a twofold benefit, by transferring computational load from the provenance store to the cache and improving query latencies because of reduced congestion resulting from the drop in network traffic. Queries through provenance-aware caches can be processed locally by fetching the necessary vertices and evaluating the query at the cache. Alternatively, the queries can be passed to the provenance store where they are evaluated with the results returned directly to the client. A third option is to use *semantic caching* [12] where the queries are partially evaluated with the remainder being shipped to the provenance store.

Caching strategies used in Web proxies [23, 24, 11, 22, 50, 49, 37] do not translate well to the provenance context. The cache managers do not exploit the relationships between the pages - that is, the link structure in the Web - to make caching decisions. The objective in the Web proxy case is to maintain a popular set of files in the cache in order to maximize hit ratios and minimize expected remote access costs for files requested but not found in the cache [49, 37]. Such caches make the assumption that each request is associated with exactly one file. In the case of provenance caches, each request involves multiple vertices in the provenance graph. Further, queries can be serviced only after all the relevant vertices are in the cache. Since each of the vertices is independent and can be shared by multiple requests it is vital for the cache to exploit the link structure to make more informed caching decisions.

Bypassing the Cache

Our caching strategy aims to load the cache with a set of vertices that will maximize the probability that an arriving request can find all the vertices it needs in the cache. Previous caching research assigns equal importance to all the objects and greedily caches those that provide the most network savings. Our approach diverges from previous caching research in two ways. First, we aim to exploit the link structure between vertices to determine the most important vertices to cache. Second, we do not attempt to just greedily cache all the vertices that improve the latency and throughput of the queries, but instead balance these goals against the amount of network traffic that the cache manager generates. Greedy caching policies can be detrimental to the performance of applications that use provenance queries in at least three possible cases. In each instance, it would be preferable for the result to bypass the cache and be returned directly to the client instead, as illustrated in Figure 5. The first such case occurs when future queries that use the retrieved vertices are expected to produce relatively little network data traffic. The second case arises when the incoming vertices are expected to

have a short lifetime in the cache. The third case materializes when different versions of the same file are rapidly being generated, invalidating the results of some provenance queries. In every case, the benefit from serving future requests from the cache will not be worth the cost of evicting other vertices to load the cache with the retrieved vertices. It is therefore preferable to force future queries to retrieve the vertices from the provenance store.

Exploiting Link Structure

We introduce the concept of a LineageRank to be employed by provenance-aware caches to determine the "importance" of vertices. The well-known PageRank [8, 7] used by Google treats a link from page A to page B as a vote from A to B. Highly linked pages are more important than pages with few links to them. Further, back links from pages with high PageRank count more than links from pages with low PageRank. LineageRank follows PageRank in that a vertex has high rank if the sum of the ranks of its back links is high. However, LineageRank also measures the depth of the vertex in the provenance graph. Thus, if two vertices have back links with the same PageRank but one is more recent or has a deeper ancestry, LineageRank can distinguish this and give that vertex a higher ranking. Unlike PageRank, LineageRank does not suffer from the rank-sink problem as it is computed over directed acyclic graphs [8].

Accounting for Network Utility

LineageRank captures as good an approximation to "usefulness" as can be derived from just the directed link structure of the provenance graph. However, it does not capture the network utility of a vertex, which is an important measure for reducing latency. We can capture this by measuring the probability of access and the quantity of vertex-related data that is actually used by a provenance query. Since the queries may filter out and aggregate partial information from the vertices, such fine-grained accounting can make a substantial difference to the efficacy of the cache eviction policy. Similarly, when calculating network utility, we assign variable benefits to vertices depending upon their frequency of access and how much of the data is actually used in query results.

5. RELATED WORK

The Web and Grid environments adopt an object caching model [22, 11, 33] in which items of variable size and cost are cached. The Greedy-Dual algorithm [50] first introduced variable fetch costs in the I/O paging model. The Greedy-Dual-Size (GDS) algorithm then adapted the technique to variable-size objects for Web caching [22, 11]. It has been found that GDS compares favorably to more heavily parameterized algorithms that handle variable sizes and retrieval costs [37, 49]. Popularity-aware versions of GDS add richer semantic information to GDS by tracking access frequency [23, 24]. Otoo [39] further improved object caching by recognizing that requests often arrive for different combinations of files, known as *bundles*, rather than just single files. The provenance model is similar to the object caching model in that items of variable size and cost need to be cached to improve the performance of queries. However, provenance queries differ from Web requests since the objects in a provenance graph are linked to each other in a structured manner that the caching mechanism must account for in order to exploit the available locality of reference. This is especially true for caching recursive lineage queries.

Centralized database management systems have also addressed scalability concerns by using caching mechanisms. Times Ten [48] and DBCache [2] use middle-tier caching to avoid bottlenecks at the central back-end server. DBProxy [4] supports structured data caching at edge servers. It caches a large number of overlapping and dynamically changing materialized views. Hierarchical, multi-level, Web-style caches have been constructed for Online Analytical Processing (OLAP) systems [25]. The primary goal in most database caching solutions is to improve application performance - that is, query latency. Reducing network traffic is considered secondary. Provenance data caching requires reduction in both latency and networking costs.

Scientific database federations [47, 32, 45], being geographically distributed and network bound, have addressed the combined issue of latency and network traffic. Bypass-Yield caches [33] tailor the Web object-caching model to the needs of wide-area distributed scientific federations to dramatically reduce bandwidth demands and improve latency. However, scientific database federations do not witness the explosion in the number of objects that occurs with provenance metadata. Relational database objects also do not exhibit the graph structure seen in provenance metadata.

The graphical structure of the provenance metadata is analogous to large-scale Web graphs as managed by Google and social networking Web sites such as FaceBook [13] and LinkedIn [31]. Google's PageRank [7, 8] algorithm has provided an effective ranking mechanism for large graph-based structures. Other link-based ranking algorithms for Web pages include Klienberg's HITS algorithm [29], which is now used at Ask.com [5]. Current research on handling Web-scale graphs does not examine the interaction of Web page ranks on cache management policies.

6. CONCLUSION

SPADE is a system for auditing fine-grained provenance in distributed environments. We described how SPADE stitches provenance graphs collected at different hosts and uses summary data structures and caching strategies to answer distributed provenance queries. SPADE adopts a decentralized model for recording provenance where the metadata is kept local to the host that generated it and is only moved on demand by a user. SPADE thus reduces the cost of expensive network transfers of metadata and provides complete control to users over their local provenance records.

7. 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] M. Altinel, Q. Luo, S. Krishnamurthy, C. Mohan, H. Pirahesh, B. G. Lindsay, H.Woo, and L. Brown, DBCache: Database caching for web application servers, *ACM SIGMOD International Conference on Management of Data*, 2002.

[3] S. Alvarez, J. Vazquez-Salceda, T. Kifor, L.Z. Varga, and S. Willmott, Applying provenance in distributed organ transplant management, *Springer Lecture Notes in Computer Science*, Vol. 4145(28), 2006.

[4] Khalil Amiri, Sanghyun Park, Renu Tewari, and Sriram Padmanabhan, DBProxy: A dynamic data cache for Web applications, *19th International Conference on Data Engineering*, 2003.

[5] Ask.com, `http://www.ask.com`

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

[7] S. Brin and L. Page, The anatomy of a large-scale hypertextual web search engine, *7th World Wide Web Conference*, 1998.

[8] S. Brin and L. Page, The PageRank Citation Ranking: Bringing Order to the Web, Stanford University Technical Report, January 29, 1998.

[9] A. Chapman and H. V. Jagadish, Issues in building practical provenance systems, *Data Engineering*, 2008.

[10] A.L. Chervenak, N. Palavalli, S. Bharathi, C. Kesselman, and R. Schwartzkopf, Performance and scalability of a replica location service, *High Performance Distributed Computing*, 2004.

[11] P. Cao and S. Irani, Cost-aware WWW proxy caching algorithms, *USENIX Symposium on Internet Technology and Systems*, 1997.

[12] S. Dar, M. J. Franklin, B. T. Jonsson, D. Srivastava, and M. Tan, Semantic data caching and replacement, *International Conference on Very Large Data Bases*, 1996.

[13] Facebook, `http://www.facebook.com`

[14] C. M. Fiduccia and R. M. Mattheyses, A linear time heuristic for improved network partitions, *19th Annual Conference on Design Automation*, 1982.

[15] 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.

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

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

[18] Ashish Gehani, Minyoung Kim, and Jian Zhang, Steps toward managing lineage metadata in Grid clusters, *Theory and Practice of Provenance* affiliated with the 7th USENIX Conference on File and Storage Technologies, 2009.

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

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

[21] 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.

[22] S. Irani, Page replacement with multi-size pages and applications to Web caching, *ACM Symposium on the Theory of Computing*, 1997.

[23] S. Jin and Z. Bestravros, Popularity-aware greedydual-size Web proxy caching, *International Conference on Distributed Computing Systems*, 2000.

[24] S. Jin and Z. Bestravros, Greedydual* Web caching algorithms: Exploiting two sources of temporal locality in Web request streams, *Computer Communications*, Vol. 24(2), 2001.

[25] P. Kalnis, W. S. Ng, B. C. Ooi, D. Papadias, and K.-L. Tan, An adaptive peer-to-peer network for distributed caching of OLAP results, *ACM SIGMOD International Conference on Management of Data*, 2002.

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

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

[28] B. W. Kernighan and S. Lin, An effficient heuristic procedure for partitioning graphs, *Bell System Technical Journal*, Vol. 49(2), 1970.

[29] Jon Kleinberg, Authoritative sources in a hyperlinked environment, *Journal of the ACM*, Vol. 46(5), 1999.

[30] 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.

[31] LinkedIn, `http://www.linkedin.com`

[32] T. Malik, A. S. Szalay, A. S. Budavri, and A. R. Thakar, SkyQuery: A WebService approach to federate databases, *Conference on Innovative Data Systems Research*, 2003.

[33] T. Malik, R. Burns, and A. Chaudhary, Bypass caching: making scientific databases good network citizens, *International Conference on Data Engineering*, 2005.

[34] 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.

[35] S. Miles, S. Munroe, M. Luck, and L. Moreau, Modeling the provenance of data in autonomous systems, *6th International Joint Conference on Autonomous Agents and Multi-agent Systems*, 2007.

[36] 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.

[37] N. Niclausse, Z. Liu, and P. Nain, A new efficient caching policy for the World Wide Web, *Workshop on Internet Server Performance*, 1998.

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

[39] E. Otoo, D. Rotem, and A. Romosan, Optimal file-bundle caching algorithms for data-Grids, *Supercomputing*, 2004.

[40] 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.

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

[42] Lineage File System, `http://crypto.stanford.edu/~cao/lineage.html`

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

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

[45] A. Szalay and J. Gray, The world-wide telescope, *Science*, Vol. 293(5537), 2001.

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

[47] A. R. Thakar, T. Budavri, T. Malik, A. S. Szalay, G. Fekete, M. Nieto-Santisteban, V. Haridas, and J. Gray, SkyQuery - A prototype distributed query and cross-matching web service for the virtual observatory, *Bulletin of the American Astronomical Society Meeting*, Vol. 34, 2002.

[48] The Times Ten Team, In-memory data management in the application tier, *International Conference on Data Engineering*, 2000.

[49] R. Wooster and M. Abrams, Proxy caching that estimates page load delays, *International WWW Conference*, 1997.

[50] N. E. Young, On-line caching as cache size varies, *Symposium on Discrete Algorithms*, 1991.