

SPADE: Support for Provenance Auditing in Distributed Environments

Ashish Gehani and Dawood Tariq

SRI International

Abstract. SPADE is an open source software infrastructure for data provenance collection and management. The underlying data model used throughout the system is graph-based, consisting of vertices and directed edges that are modeled after the node and relationship types described in the Open Provenance Model. The system has been designed to decouple the collection, storage, and querying of provenance metadata. At its core is a novel provenance kernel that mediates between the producers and consumers of provenance information, and handles the persistent storage of records. It operates as a service, peering with remote instances to enable distributed provenance queries. The provenance kernel on each host handles the buffering, filtering, and multiplexing of incoming metadata from multiple sources, including the operating system, applications, and manual curation. Provenance elements can be located locally with queries that use wildcard, fuzzy, proximity, range, and Boolean operators. Ancestor and descendant queries are transparently propagated across hosts until a terminating expression is satisfied, while distributed path queries are accelerated with provenance sketches.

1 Introduction

The origin of SPADE [59] can be traced to a discussion in 2006 with a member of the BaBar [5] project at SLAC [58]. BaBar consists of more than 500 physicists and engineers, maintains petabytes of information in databases, and processes large volumes of data using computational Grids that consist of computer clusters in multiple administrative domains. One conclusion from the discussion was that despite the long history of research in distributed computing, the issue of how to ascertain the security of data in Grid environments (with hundreds of users from scores of independent organizations) was still open to debate.

Extant filesystems reported minimal information about the history of stored data, leaving the task of maintaining such records to individual applications. While knowledge of lineage would allow the trustworthiness of data to be ascertained, support to answer such queries was limited (typically to determining the time and user involved in the original creation and last modification of a file). The gap provided the impetus to create SPADE in 2008 as a distributed service for collecting, certifying, and querying the provenance of Grid data [56].

The first version (SPADEv1) tackled a combination of fundamental challenges, including provenance growth and verification latency, as well as practical concerns, such as the need to support legacy environments. SPADEv1 used selective provenance replication to increase distributed availability while limiting the storage overhead [15].

It aggregated, reordered, and query-specifically pruned provenance elements to improve latency and reliability when verifying responses [18], and embedded provenance *witnesses* (precursors of *sketches* [17,39]) as hints to reduce extraneous remote connections in distributed provenance queries [18].

To collect provenance without modifying applications or the operating system, events from a user space filesystem [50] were fused with process-related information from `/proc` (on Linux). Unmodified applications could ensure that a file’s provenance was transparently transferred across network connections. This was accomplished by appending the provenance to the content if the filename was suitably augmented when the file was opened for reading, and analogously extracting and recording the appended provenance at the other end if the file was saved with an augmented filename [16].

In late 2009, the NIGHTINGALE project [45] began experimental use of SPADEv1. NIGHTINGALE involved experts from 15 universities and corporations concurrently developing parts of a speech technology toolchain that processed terabytes of data on hundreds of computers. We expected that the provenance of intermediate outputs would be used to optimize the subsequent steps in workflows. In practice, application-generated metadata was maintained for this. Instead, SPADEv1 was used to locate bottlenecks in distributed workflows by adding support to capture input and output attributes and recording them in the provenance. It was also actively used to identify code and data dependencies when releasing new versions of the toolchain.

Given the number of institutions involved, we anticipated that provenance certification would be widely employed, but it was not. We learned that SPADEv1’s design meant certification was finer-grained than warranted in many situations. Similarly, the architecture imposed a high overhead for incorporating additional provenance attributes, experimenting with novel storage and indexing models, and handling provenance from diverse sources. This motivated a redesign in 2010.

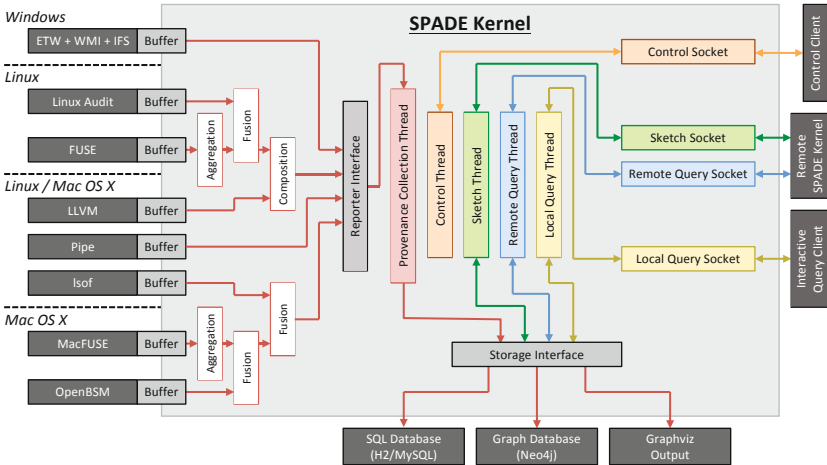





Fig. 1. SPADEv2 has a cross-platform kernel that decouples the collection, storage, and querying of provenance metadata derived from applications, operating systems, and network activity

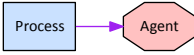
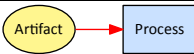


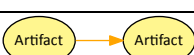
SPADEv2 is the second generation of our data provenance collection, management, and analysis software infrastructure. The underlying data model used throughout the system is graph-based, consisting of vertices and directed edges, each of which can be labeled with an arbitrary number of annotations (in the form of key-value pairs). These annotations can be used to embed the domain-specific semantics of the provenance.

The system has been completely re-architected to decouple the production, storage, and utilization of provenance metadata, as illustrated in Figure 1. At its core is a novel *provenance kernel* that mediates between the producers and consumers of provenance information, and handles the persistent storage of records. The kernel handles buffering, filtering, and multiplexing incoming metadata from multiple provenance sources. It can be configured to commit the elements to multiple databases, and responds to concurrent queries from local and remote clients. The kernel also supports modules that operate on the stream of provenance graph elements, allowing the aggregation, fusion, and composition of provenance elements to be customized by a series of filters.

SPADEv2 supports the Open Provenance Model [42,47] and includes controlling *Agent*, executing *Process*, and data *Artifact* node types, as well as dependency types that relate which process *wasControlledBy* which agent, which artifact *wasGeneratedBy* which process, which process *used* which artifact, which process *wasTriggeredBy* which other process, and which artifact *wasDerivedFrom* which other artifact. Table 1 illustrates how each of these nodes and dependencies represented.

Table 1. SPADE can emit provenance graphs in Graphviz [21] syntax with Open Provenance Model (OPM) semantics. The encoding of the provenance elements is summarized here. Provenance domain semantics are added to the vertices and edges as annotations.

OPM Node	Node Encoding	Graph Representation
<i>Agent</i> vertex	Red octagon	
<i>Process</i> vertex	Blue rectangle	
<i>Artifact</i> vertex	Yellow ellipse	

OPM Dependency	Dependency Encoding	Graph Representation
<i>WasControlledBy</i> edge	Purple arrow (from <i>Process</i> to <i>Agent</i>)	
<i>WasGeneratedBy</i> edge	Red arrow (from <i>Artifact</i> to <i>Process</i>)	
<i>Used</i> edge	Green arrow (from <i>Process</i> to <i>Artifact</i>)	
<i>WasTriggeredBy</i> edge	Blue arrow (from <i>Process</i> to <i>Process</i>)	
<i>WasDerivedFrom</i> edge	Yellow arrow (from <i>Artifact</i> to <i>Artifact</i>)	

2 Provenance Kernel

The kernel is designed to be extensible in four ways. A *reporter* can be added to collect provenance activity about a new domain of interest. A *filter* can be inserted to perform a new transformation on provenance events in the kernel. A *storage* system can be introduced to record provenance in a new format. A *sketch* can be used to optimize the distributed querying. The kernel is written in Java and uses a combination of the runtime's dynamic classloading and abstract classes to facilitate the concurrent addition and removal of reporters, filters, sketches, and storage through the *control client*. A different abstract class defines the framework for each type of extension and how it interfaces with the kernel.

The control client maintains a history of commands, and allows a combination of extensions to be saved to or loaded from a configuration file. When the control client shuts the kernel down, callbacks are invoked in all extensions to shut them down gracefully (flushing buffered data as necessary) and the kernel's configuration is saved so it can be automatically loaded the next time it runs.

When SPADEv2 is activated on a computer, the provenance kernel is launched as a daemon that runs in the background. Its functionality can be invoked on demand with low latency and imposes low overhead when not in active use. Initially, the kernel is blind to provenance reporting, deaf to control and query clients, and mute about events it has learned of. It uses saved configuration information, if available, to spawn threads for each of the tasks described in Table 2.

Table 2. The SPADE kernel is multi-threaded to allow provenance reporting, local and remote querying, and reconfiguration of the kernel to operate concurrently

Thread	Service Provided
<i>Provenance Collection</i>	Reporters queue events programmatically in buffers that must then be emptied by a thread in the kernel. The thread extracts the events, filters them, and commits them to local storage.
<i>Remote Queries</i>	Each kernel is implicitly part of a peer-to-peer provenance overlay, with a thread handling provenance queries from remote kernels.
<i>Sketches</i>	A kernel on another computer may need a provenance sketch to optimize its distributed queries. Such requests are processed through a separate port and thread.
<i>Local Queries</i>	Interactive use with query clients requires extra information, including user prompts and error reporting, to be multiplexed with the responses to queries, and is therefore handled by a thread that is distinct from the one for remote queries.
<i>Kernel Control</i>	Since the kernel operates as a background system service, it uses a thread to listen for connections on a control port that then processes commands received from the control client.

When a provenance event occurs, SPADEv1 blocked until the activity had been completely recorded. This had the advantage that the provenance records were always synchronized with the state of the system (from which provenance events were derived). However, it had the disadvantage that application performance was adversely impacted by the latency introduced during input and output operations. The design choice had been made to support workflows that used the provenance of intermediate data to decide subsequent steps. In practice, the tight coupling was seldom necessary.

The SPADEv2 kernel provides a non-blocking interface for reporting provenance events. While this ensures that the monitoring overhead for provenance collection is minimal, it introduces a new concern. In scenarios where the rate at which provenance is being reported varies significantly, there are periods when the kernel cannot process events at the rate that they are arriving. In this situation, some events are lost. To mitigate this, SPADEv2 creates separate *buffers* for each provenance reporter to enqueue events into. A thread in the kernel then dequeues events when the load permits, processes the events through any configured filters, and sends the results to persistent storage.

3 Generating Metadata

Although users do maintain provenance records manually (in the form of scientific laboratory notebooks, for example), automating the generation and collection of provenance metadata substantially reduces their burden, improves reproducibility, aids in debugging, and increases the utility of their data to other researchers. To facilitate this, SPADEv2 includes a number of reporters that transparently transform computational activity into provenance events that are sent to the kernel.

Each reporter utilizes the same interface to the kernel, abstracted in Figure 2. This holds regardless of whether the provenance elements are manually curated, application emitted, logged by a workflow engine, or from the operating system’s audit trail. The domain semantics are captured as annotations on the vertices and edges.

```
putVertex(Agent a);
putVertex(Process p);
putVertex(Artifact a);

putEdge(Used u);
putEdge(WasControlledBy wcb);
putEdge(WasDerivedFrom wdf);
putEdge(WasGeneratedBy wgb);
putEdge(WasTriggeredBy wtb);
```

Fig. 2. A reporter emits a provenance element by calling the appropriate function, which queues it in a buffer. The kernel multiplexes elements from the buffers of all reporters.

3.1 Operating System Provenance

The advantages of collecting data provenance at the operating system level are that it provides a broad view of activity across the computer and that it does not require applications to be modified. The approach has a number of limitations, including the fact that this may provide too much extraneous information and not enough detail about particular applications that are of interest. A significant consideration for software maintainability is how the system activity is obtained. Implementing a kernel module or modifying system libraries requires a substantial investment in adapting the collection mechanism to each currently available and future version of the operating system.

An alternative approach relies on utilizing the auditing mechanisms of each operating system, which typically have stable programming interfaces across operating system versions. The disadvantage of the technique is that it is limited to the information exposed in the audit trail, which does not include records of interprocess communication through shared memory, graphical user interface events, or keyboard input. Nevertheless, the provenance collected suffices for characterizing the batch computing workloads that are the staple of scientific computing workflows. In particular, this includes the process's name, owner, group, parent, host, creation time, command line, environment variables, and a file's name, path, host, size, and modification time. The types of provenance collected at the operating system level are summarized in Table 3.

Linux (System-wide): An audit trail is needed for the Common Criteria certification of systems used by U.S. Government agencies. Linux vendors interested in sales to this market contributed kernel changes to monitor activity across the entire host and generate corresponding audit events. These are accessible through a Unix socket (*/var/run/auditd.events*) after activating a system service (*auditd*) with an appropriate plug-in. SPADEv2's Linux reporter configures the audit system to generate records for *exec()*, *fork()*, *clone()*, *exit()*, *open()*, *close()*, *read()*, *write()*, *clone()*, *truncate()*, and *rename()* system calls.

Since Java does not support reading from Unix sockets, a utility written in C serves as a bridge. The audit records are then parsed in the Java component of the reporter. Reporting *read()* and *write()* events poses two challenges. First, the Linux audit records contain only a file descriptor, so a mapping between descriptors and filenames has to be built using information from *open()* calls. Second, reporting *read()* and *write()* events would provide enough provenance metadata that system responsiveness would noticeably degrade. Consequently, these two calls are not reported. Instead, the flags of *open()* calls are used to infer whether a process is reading or writing a file. If detailed input and output records are needed, the alternative Linux reporter that focuses on selected filesystem activity can be utilized.

Process-related information is obtained from two sources. When a system call occurs, the kernel generates an audit record. The reporter extracts the process identifier from this record. This identifier is then used to obtain further details about the process from the Linux */proc* filesystem, if available. Since the audit record is created in the kernel but used in user space, it may be reporting the action of a process that has already terminated, with no corresponding information available under */proc*. In this case, other elements of the audit record (such as the name, owner, and group of the process) are used. On the surface, the approach employed appears to introduce a *time-of-check-to-time-of-use* race condition. However, this is not the case since process identifiers are allocated serially. A problem would manifest only if the process identifier value wrapped through the entire possible range within the time window between the check and use.

We found that network-related system calls (such as *connect()* and *accept()*) report only the remote IP address and port information. The source IP address and port are not recorded, preventing connections from being completely disambiguated. This weakness is partially addressed by the Network reporter.

Linux (Selected activity): The hooks and module needed to support FUSE [14] user space filesystems are present in all Linux kernels, starting with version 2.6.14. In particular, interposition can be limited to file activity in a specific directory, eliminating the monitoring of calls to files outside the subtree. This allows detailed provenance to be recorded with low overhead for workloads that are localized to a single subtree (as is the case with many scientific and engineering applications), including annotations on *used* and *wasGeneratedBy* edges for the time spent in *read()* and *write()* calls.

The reporter includes C code that is linked against the FUSE shared library that handles communication with the kernel. This code is invoked when *read()*, *write()*, *rename()*, *truncate()*, *link()*, *symlink()*, *readlink()*, and *unlink()* filesystem calls occur, and the arguments to each call are passed via the Java Native Interface (JNI) [31] to Java code that transforms the filesystem event into appropriate provenance elements. The identifier of the process that made the filesystem call is used to extract more information about the process from the Linux `/proc` filesystem. Since the system call is blocked during this step, the process record will always be present and the information extracted will be current and accurate.

If a process does not interact with the filesystem, it will not trigger a FUSE event. In this situation, no information would be collected about the process. To mitigate this limitation, information about all ancestor processes is also extracted and added to the provenance record. In this context, it is worth noting that when a process exits, the Linux kernel changes the parent of all child processes to the parent of the exiting process. This can result in multiple (consistent) accounts of the lineage of a single process.

Android (System-wide): Google’s mobile device platform, Android, uses a Linux kernel. We therefore assumed that the audit-based Linux reporter would be usable for collecting data provenance. However, a number of challenges arose, including the absence of audit code in the Linux kernel for ARM processors, and the audit daemon *auditd*’s dependence on *glibc* functions not present in Android’s replacement *bionic* library. Using our patch (that is now part of Linux kernel 3.3) and modified audit utilities, SPADEv2 can collect Android provenance. It is worth noting that this is lower-level activity than would be generated by a reporter that instrumented Android’s Dalvik virtual machine. Interactions between applications are captured using the transaction log (in `/proc`) of the Binder inter-process communication mechanism.

Mac OS X (System-wide): The Basic Security Module (BSM) system was designed by Sun Microsystems. It includes a framework for generating, accessing, and parsing audit records in a documented format. Apple had it ported to Mac OS X to obtain Common Criteria certification. The open source version is maintained as OpenBSM [46]. The Mac OS X kernel reports system events in real time. A process with sufficient privilege can access the resulting records by reading the named pipe `/dev/auditpipe`. Using an *ioctl()* system call, the pipe can be configured to specify which system events are of interest. The system-wide Mac OS X reporter consists of C code that runs with *setuid*, configures the pipe, and then forwards the output to unprivileged Java code where it is parsed and used to generate appropriate provenance events. The set of system calls monitored includes *fork()*, *exit()*, *kill()*, *read()*, *write()*, *create()*, and *rename()*.

Each audit record includes the identifier of the process responsible for the action. Since OpenBSM can be configured to record the command line arguments and

environment variables when a process is invoked, in principle the audit records should have sufficient process-related information. However, the OpenBSM subsystem on OS X Snow Leopard does not audit the *spawn()* system call, which is used by the *Finder* to launch applications. Therefore, even though *fork()* and *exec()* calls are audited, a significant amount of process-related provenance is lost (since processes started with *spawn()* are not observed). To address this limitation, the reporter extracts the process identifier and obtains further information about the process with the *ps* utility. This approach cannot collect information about a short-lived process that may have terminated before *ps* was invoked. It is worth noting that the serial allocation of process identifiers ensures that information about the wrong process is never collected.

Since system-wide activity is monitored, only the first read from and write to a file by a process are recorded to minimize the performance overhead. The alternative Mac OS X reporter (that focuses on selected filesystem activity) can record details about reads and writes, should that level of detail be needed. Further, when network connections occur, the BSM records generated have invalid IP addresses on OS X Snow Leopard and OS X Lion, preventing the construction of provenance artifacts to represent the connections. The Network reporter attempts to address this weakness.

Mac OS X (Selected activity): An alternative reporter for Mac OS X that leverages the MacFUSE [37] user space filesystem was developed to limit provenance collection to a subtree in the filesystem. This facilitates managing the overhead associated with recording *read()* and *write()* calls. The reporter contains C code that is called when *read()*, *write()*, *rename()*, *link()*, *symlink()*, *readlink()*, and *unlink()* calls occur. Each invocation results in a call through JNI to Java code that converts the filesystem event into a corresponding provenance event.

Information about the process that made a filesystem call is obtained with the *ps* utility. In contrast to the system-wide reporter, where the invocation of *ps* is not synchronized with the system call being audited, here the filesystem call blocks during the invocation of *ps*, ensuring that metadata is collected even for short-lived processes.

As with the Linux FUSE-based reporter, a process that does not interact with the filesystem does not trigger the collection of its provenance. This prevents descendant processes from being linked to ancestor processes, and creates a problem in practice with gaps in provenance chains. To mitigate this issue, when information is collected about a process, records are constructed for all known ancestor processes as well.

Though MacFUSE requires administrator privileges to be installed (since it uses a Mac OS X kernel extension), it is used by numerous other software packages and may already be installed and available on the user's system. This is of particular utility in situations where the user does not have permission to install a *setuid* program (as is needed for the system-wide Mac OS X reporter).

Windows (System-wide): Microsoft's Event Tracing for Windows (ETW) [10] framework allows application developers to use system-level information for debugging and performance analysis. Since ETW provides a documented interface for collecting information about operating system activity, we used it to generate provenance records. However, ETW provides process identifiers only in event descriptions, necessitating the use of Microsoft's Windows Management Instrumentation (WMI) [66] framework

to obtain details such as a process’s name, binary location, creation time, and command line.

When ETW generates file events, it records the associated filenames internally but does not make them available until the end of the tracing session. This prevents the online generation of provenance artifacts. Microsoft’s Windows Driver Kit (WDK) [63] includes the Installable File System (IFS) Kit [28], which can be used to write filters that intercede on filesystem calls. We developed an IFS filter to monitor file creation, reads, and writes.

Consequently, our initial Windows reporter consisted of C++ code that interfaced with the ETW, WMI, and IFS subsystems. The C++ code has been replaced by an invocation of the *Process Monitor* tool [49], which interfaces with the Windows subsystems and emits a log. The Windows reporter now parses the events in the log and transforms them into provenance elements that are sent to the SPADEv2 kernel.

The approach of relying on an external tool to collect system activity resolved three issues. First, it allows the reporter to run on all versions of Windows released after 2000. In contrast, the initial reporter supported only a single version of the operating system since the programming interfaces of ETW and WMI differ across releases of Windows. Second, it eliminates the need for IFS driver signing since Process Monitor has a signed kernel driver. Third, it eliminates a dependency on Microsoft source code that could not be redistributed with SPADEv2 due to an incompatible license.

Table 3. In this summary of operating system provenance reporting, a check mark in a cell indicates that the operation listed in the column is recorded by the reporter listed in the row. The last row depicts the Open Provenance Model semantics of the operation.

	Open File for Reading	Open File for Writing	Read File	Write File	Rename File	Create Link	Transmit Data	Receive Data	Create Process
Linux	✓	✓							✓
Mac	✓	✓			✓				✓
Windows			✓	✓			✓	✓	✓
Selected			✓	✓	✓	✓			
Network							✓	✓	
Provenance Semantics									

Network: SPADE aims to support provenance queries about distributed computations. Whereas SPADEv1 was limited to noting the relationship between a source and

destination file when a remote copy occurred, SPADEv2 explicitly models a network connection as a pair of *network artifacts* connected by *used* and *wasGeneratedBy* edges.

Network artifacts (depicted by green diamonds in Table 3) are distinguished by the property that each endpoint can independently construct the same artifact without explicit coordination. This allows the complete decentralization of provenance collection in distributed systems while still ensuring that subgraphs from different hosts can be reassembled into a coherent reconstruction of distributed data provenance. SPADEv2 implements network artifacts with this property combining the time the connection was initiated with the IP addresses and TCP or UDP ports of the two endpoints.

None of the Linux or Mac OS X reporters have access to correct source and destination IP address and TCP or UDP port information. Consequently, a separate reporter uses the *Isof* [35] utility in repeat mode to poll the operating system and periodically retrieve a list of recent connections. These are transformed into provenance semantics and then sent to the SPADEv2 kernel. While the reporter is not asynchronously notified of new connections, it is able to report network provenance metadata within a second of the connection's occurrence. The synchronous inspection of network activity means that short-lived connections are unlikely to be reported.

3.2 Application Provenance

An advantage of collecting data provenance from the operating system, as described in Section 3.1, is that applications can be monitored without any provenance-specific modifications. However, instrumentation at this level of abstraction results in an operating system process being modeled as a monolithic entity. Since intra-process data flow (such as memory reads and writes) is not recorded, internal application-level dependencies cannot be differentiated. Further, the provenance semantics of interest in an application may manifest at a higher level of abstraction than operating system interfaces. SPADEv2 includes two types of support for collecting application-level data provenance on both Linux and Mac OS X.

Domain-Specific Language: In late 2010, scientists at SRI were managing large volumes of mass spectroscope data. They were interested in using SPADEv2 to track the computational manipulation of the records. Since the steps were performed in MATLAB [40], we needed a mechanism to communicate provenance information from an external application to the SPADEv2 kernel.

One possible approach would have been to create a dynamically linked library with functions for reporting provenance metadata, similar to Harvard's Core Provenance Library [38]. We adopted an alternative approach for a number of reasons. First, the target application's source would have to be available, which is not the case for commercial applications such as MATLAB. Second, determining where to insert the calls to the provenance reporting functions would require extensive study of the target application's codebase. Third, the library would reside in the address space of the target application, leaving the issue of communicating the metadata to the SPADEv2 kernel unresolved.

Instead, we developed a reporter that creates a named pipe, continuously reads from it, parses any information it retrieves, and constructs appropriate provenance elements that are then sent to the SPADEv2 kernel. Provenance metadata can be sent to the reporter by any source that can write to a named pipe, including external applications

and users interested in manually adding provenance records. The provenance metadata must be stated in a simple OPM-inspired domain-specific language. The corresponding context-free grammar is shown in Backus-Naur Form in Figure 3.

```

<provenance> ::= <provenance> <element> | <element>
  <element> ::= <node> | <dependency>
    <node> ::= <node-type> <node-id> <annotation-list>
  <node-type> ::= type: <vertex-type>
<vertex-type> ::= Agent | Process | Artifact
  <node-id> ::= id: <vertex-id>
  <vertex-id> ::= <unique-identifier>
<annotation-list> ::= <annotation-list> <annotation> | <annotation>
  <annotation> ::= <key> : <value>
  <dependency> ::= <dependency-type> <start-node> <end-node>
    <annotation-list>
<dependency-type> ::= type: <edge-type>
  <edge-type> ::= WasControlledBy | WasGeneratedBy | Used |
    WasTriggeredBy | WasDerivedFrom
  <start-node> ::= from: <vertex-id>
  <end-node> ::= to: <vertex-id>
    
```

Fig. 3. The grammar for the domain-specific language that can be used by external applications to report Open Provenance Model metadata to the SPADEv2 kernel

Compiler-Based Instrumentation: Manually instrumenting an application to emit provenance metadata requires a substantial effort. This becomes decreasingly tenable as the scale of the software system increases. To address this, we developed LLVM-based [34] compiler support to automate the process of instrumenting an application to emit intra-process provenance information at function call granularity [61].

In many instances, the function call level of abstraction corresponds to what the user is interested in. However, this may still result in reporting far more information than the user is interested in since every function call will be reported. To avoid overwhelming the user with extraneous information, we allow only the functions that are of interest to be specified. The program sources are statically analyzed to obtain the application’s call graph, which is then traversed in a reverse reachability analysis to identify which functions should be reported. Provenance metadata about all other functions is discarded.

An advantage of recording provenance at the finer application function call level is that it reduces the process-level *dependency aliasing* that results when collecting provenance using system calls. For example, the provenance of data transmitted over a network connection includes all the files read until that point by the server, if provenance is collected at the operating system level. If individual threads read different files and sent them to distinct network connections, well-structured code would allow function call level provenance to distinguish the dependencies.

In practice, users are interested in the values of arguments to function calls. However, providing meaningful information about the arguments requires knowledge of their types, which is often lost in the process of compiling from the source language

to LLVM's intermediate representation, *bitcode*. Since provenance instrumentation is inserted in the bitcode, only pointers to such values can be reported.

4 Persistent Storage

SPADEv1 used a relational database to store the provenance metadata as it was being collected. This meant that provenance collection could proceed only at the rate that transactions could be committed to the database. Graph queries were constructed as SQL queries, with repeated self-joins to compute the transitive closures necessary to answer path queries. The addition of new attributes resulted in changes to the relational schema. Each of these contributed to performance degrading as the provenance graphs grew in size. For users collecting large volumes of provenance metadata and primarily initiating graph queries, a graph database seemed to be a better option.

Despite the limitations of storing provenance in SQL databases, it remained an attractive option for some users. This is the case if the quantity of provenance is smaller (as is the case when provenance is collected from a source reporting it at a higher level of abstraction or over a shorter span of time), the query workload is well supported by relational operators, or the user has SQL infrastructure and experience that can be leveraged.

SPADEv2 allows arbitrary types of persistent storage to be used as a back end. It does this by defining an abstract storage interface. An adapter for a back end implements the subset of the storage interface that the repository can support. The query interface forwards requests without interpreting them. This allows the SPADEv2 kernel to utilize the native query capabilities of each type of storage.

Neo4j: Neo4j [44] is a high-performance cross-platform graph database with support for transactions. It allows vertices and edges to be typed and annotated, provides a rich set of graph querying functionality, and incorporates Apache Lucene [36] indexing of the graph data. Lucene provides Boolean, wildcard, fuzzy, proximity, range, boosting, and grouping operators for flexible querying. Neo4j is the default database used by SPADEv2.

SQL: To facilitate storing provenance in SQL databases, SPADEv2 includes a JDBC-based [30] storage adapter. By default, the SQL adapter uses the cross-platform embedded relational database, H2 [27]. However, it can also use an alternative JDBC-compliant database, such as MySQL [43], by specifying the driver at activation. The adapter supports recording provenance elements in vertex and edge tables, and SQL queries over these tables, but does not implement graph functionality in the storage interface, such as path queries.

After the SQL storage has been loaded, every provenance node is added as a new row in the table of vertices. When an annotation has a key that has not previously been observed, the table's schema is extended with a new column for the key. The value in an annotation is stored in the cell corresponding to the row of the vertex and the column of the key. Incoming dependencies are similarly added to the table of edges, with the schema continuously evolved to handle new keys in annotations on edges.

Graphviz: Graphviz [21] was created in 1988 by AT&T Research to facilitate graph visualization. Over the years, visualization and analysis tools have adopted the Graphviz DOT language for storing and manipulating graph data. Once the SPADEv2 Graphviz storage is loaded, every provenance element and dependency is output in DOT syntax to a file. This file can then be used with Graphviz tools that employ a variety of graph layout algorithms, as well as a wide range of other graph visualization applications. Querying is not supported by the Graphviz storage.

5 Filtering

Automated provenance collection can result in large volumes of metadata. As more information is generated and stored, both the precision and performance of queries start to degrade. One strategy to address this is to abstract the information and filter out elements if possible. In addition, when provenance is collected from multiple sources, normalizing and reconciling the streams before they are committed to persistent storage can improve subsequent query precision and performance. Therefore, the SPADEv2 kernel supports aggregation, fusion, and composition *filters* that can be used to normalize and reconcile provenance elements [19].

Temporal Aggregation: In environments where numerous low-level events are generated, *aggregation* can mitigate information overload. For example, the provenance elements of a group of readings that are close in value and from a network of sensors can be combined into one provenance element that describes the set of sensors and the value range. An analogous incentive is present for the provenance of data from cyber-physical systems such as SCADA process controllers, but with aggregation occurring over the time variable instead of the spatial one of sensor networks. When the readings do not change, the provenance elements can be aggregated into one that includes the interval of invariance. SPADEv2 includes a filter with the same motif for operating system provenance, where the provenance of a non-interleaved sequence of reads or writes can be replaced with a single provenance element that has an annotation added to describe the start and end points of the sequence.

Multi-source Fusion: When two or more reporters report provenance about the same phenomena, the semantics of the reported events may overlap. If the reporters operate at similar levels of abstraction, *fusion* allows distinct provenance elements (generated by different reporters) to be combined to provide a more complete representation of the same underlying phenomenon. As an example, reporters that capture events across the whole operating system typically report with coarse temporal granularity. A reporter that focuses on selected filesystem activity can track and add annotations about the exact quantity of time spent for each `read()` or `write()` operation for application profiling. SPADEv2 includes a filter to reconcile the two perspectives through fusion keyed on a common key (such as the process identifier), allowing a single view of operating system activity with the input and output times added to the appropriate *used* and *wasGeneratedBy* edges.

Cross-Layer Composition: When reporters operate at different levels of abstraction, *composition* can relate the activity with an *isAbstractedBy* edge. For example, the *Process* vertex for a function call can have an *isAbstractedBy* edge to the operating system

Process vertex of the application in which the call occurred. Such edges can be used to connect provenance from the LLVM-based intra-process level provenance reporter described in Section 3.2 and an operating system-level provenance reporter described in Section 3.1.

6 Evaluation

To evaluate the performance of SPADEv2, we measured the overhead of collecting provenance while building and running the Apache Web server [4] and running the BLAST genome sequence alignment tool. SPADEv2 was run in the background on Mac OS X 10.6.8, Linux Fedora 17, and Windows 7 with system-wide reporters. All experiments were performed on a 2.4 GHz Intel Core i5 machine with 4 GB of memory.

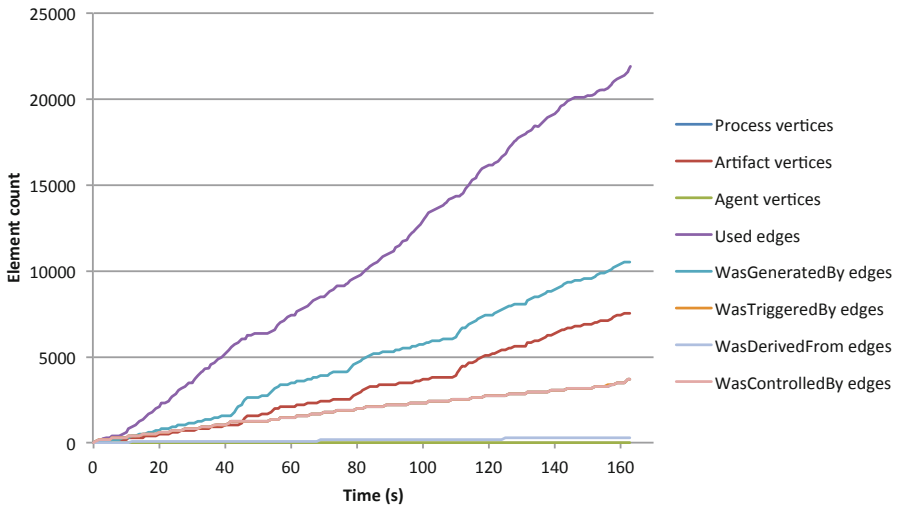


Fig. 4. Number of provenance elements generated over time during the build process of the Apache Web server on Linux is reported

To provide insight into the rate at which provenance elements are generated, Figure 4 shows the count of different types of provenance elements as they are emitted while the Apache Web server is being built. The Linux reporter for selected activity was used to collect the provenance metadata.

Figure 5 reports the time to build the Apache Web server on Windows, Mac OS X, and Linux. This time is reported for an unmodified system as well as one that has been augmented with SPADEv2 to collect provenance with a system-wide reporter. The comparison is intended to provide an understanding of the overhead incurred by collecting provenance during a compute-intensive task. The Windows reporter imposes a 53% overhead during the Apache build, presumably because a wide range of system calls are invoked and audited. On Mac OS X and Linux, the overhead was less than 10% and 5%, respectively.

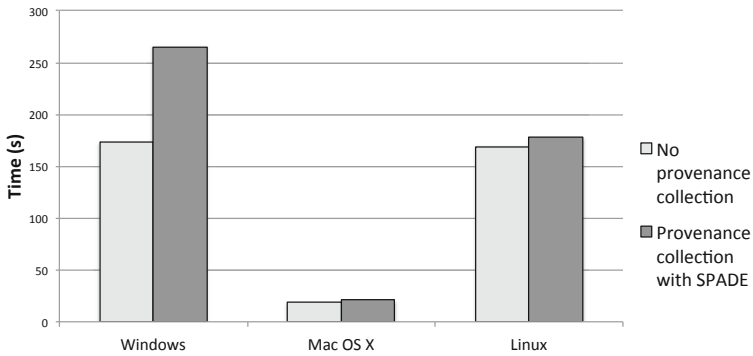


Fig. 5. Time to build the Apache Web server on multiple operating systems is measured here

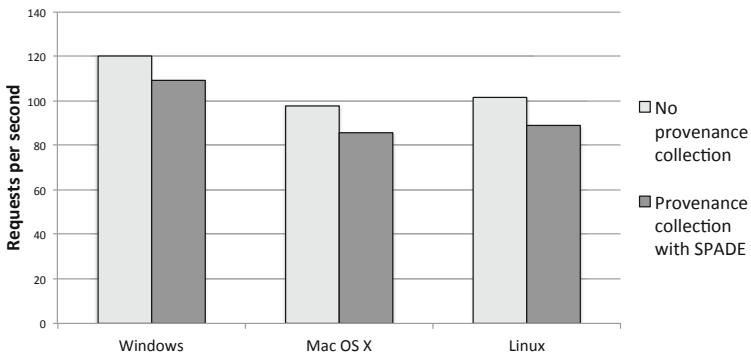


Fig. 6. Number of requests that can be handled by the Apache Web server on Windows, Mac OS X, and Linux is measured to understand the overhead of collecting provenance during normal operation

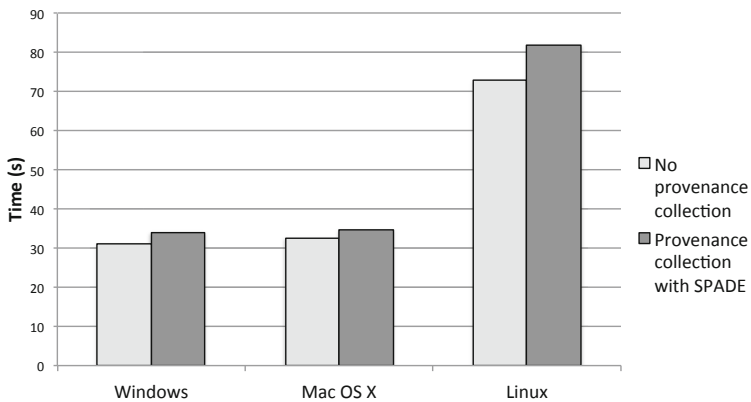


Fig. 7. The time to run the BLAST genome sequence alignment tool on multiple operating systems is measured to understand the overhead of collecting provenance during a heavy workload

To understand the overhead of collecting provenance when a service is running, the Apache Web server was run on Windows, Mac OS X, and Linux. In each case, the rate at which the Web server is able to handle requests is reported in Figure 6. This is done for both an unmodified system as well as one where SPADEv2 is running and collecting provenance with a system-wide reporter. When provenance is being collected, the performance of Apache drops by 9% on Windows and 12% on Mac OS X and Linux.

To estimate the overhead of collecting provenance when using a scientific application, we ran the BLAST [3] genome sequence alignment tool with the *influenza* data set [29] from the National Institutes of Health. Figure 7 shows that when provenance is being collected, the overhead imposed on Windows, Mac OS X, and Linux is 9%, 6%, and 12%, respectively. Since the tool invokes a limited number of system calls, the difference in overheads is likely an artifact of the set of calls being utilized.

7 Related Work

Data provenance has a range of applications. HP SRC's Vesta [24] uses it to make software builds incremental and repeatable. Lineage File System [55] records the input files, command line options, and output files when a program is executed. Its records are stored in a database that can be queried to reconstruct the lineage of a file. Provenance-Aware Storage System [51] augments this with details of the software and hardware environment. The provenance of Datalog programs has been tracked with semi-rings [22].

Several Grid environments account for data provenance in their design. *myGrid* [67] with Taverna [2] allows biologists to add application-level annotations of the data's provenance. This is then stored in the user's repository, although it does not enable other users of the data to determine its provenance. The Provenance Aware Service Oriented Architecture (PASOA) project [41] arranges for data transformations to be reported to a central provenance service [60], which can be queried by other users as well. The more recent ES3 model [13] extracts provenance information automatically from arbitrary applications by monitoring their interactions with their execution environment and logs them to a customized database. While ES3 logs events at a more abstract level than PASS, it follows the same centralized model of metadata logging. The approach ensures that the provenance does not have to be replicated. However, in the event that the metadata is heavily accessed, the latency of performing remote lookups can degrade application performance.

A number of distributed systems have been built to help scientists track their data. Chimera [11] allows a user to define a workflow, consisting of data sets and transformation scripts. The system then tracks invocations, annotating the output with information about the runtime environment. CMCS [48] is a toolkit for chemists to manage experimental data derived from fields like combustion research. It is built atop WebDAV [64], a Web server extension that allows clients to modify data on the server. ESSW [12] is a data storage system for earth scientists. If a script writer uses its libraries and templates, the system will track lineage so that errors can be tracked back to maintain the quality of data sets. A number of systems track the provenance of database elements, including Trio [65], DBNotes [6], and Perm [20]. Trio also allows the source of uncertainty to be traced. VisTrails [57] tracks the provenance of visualization workflows. Bose and

Frew's survey [7] identifies a number of other projects that aid in retrieving the lineage of scientific data.

PASS describes global naming, indexing, and querying in the context of sensor data [53]. PA-NFS [52] enhances NFS to record provenance in local area networks. Harvard's PQL [26] describes a new language for querying provenance and leverages the query optimization principles of semi-structured databases. However, it does not consider distributed naming explicitly. SPADEv2 addresses the issue by using storage identifiers for provenance vertices that are unique to a host and requiring distributed provenance queries to disambiguate vertices by referring to them by the host on which the vertex was generated as well as the identifier local to that host.

ExSPAN [68] allows the exploration of provenance in networked systems and extends traditional relational models for storing and querying provenance metadata, while SPADEv2 supports both graph and relational database storage and querying. PASS has explored the use of clouds [53,54]. Probase [1] uses Hbase, an open-source implementation of Google's BigTable [9], to store and query scientific workflow provenance. IBM researchers have proposed a provenance index that improves the execution of forward and backward provenance queries [32]. A number of efforts, including SPADEv2, have recently considered how to compress provenance [68,39]. Query optimization techniques on compressed provenance data has also been examined [25].

The Open Provenance Model (OPM) [42,47] facilitates interoperability between systems by providing a common model for describing provenance. Several projects provide OPM-compliant provenance, including SPADEv2 [59], PASS [52], VisTrails [8], and Tupelo [62]. An OPM profile [23] provides conventions for modeling distributed aspects of provenance, such as transactions. However, query interoperability and global naming are not addressed.

8 Conclusion

SPADEv2 provides a cross-platform distributed data provenance collection, filtration, storage, and querying service. It defines reporters that can be inserted between an application and the operating system, between functions of an application, or at arbitrary user-defined interfaces. Once inserted, the infrastructure operates as middleware, monitoring the targeted applications and enabling provenance analysis for a variety of purposes, including facilitating experiment reproducibility, distributed debugging, and determining dependencies when sharing data and code. We empirically compared and reported the cost of running applications with and without the middleware.

Acknowledgments. We thank Maisem Ali, Basim Baig, Nathaniel Husted, Minyoung Kim, Florent Kirchner, Hasnain Lakhani, Tanu Malik, Ian Mason, Ligia Nistor, Sharjeel Qureshi, Aditya Rajgarhia, Hassen Saïdi, Fareed Zaffar, and Jian Zhang for their contributions.

This material is based upon work supported by the National Science Foundation under Grants OCI-0722068 and IIS-1116414. 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. Abraham, J., Brazier, P., Chebotko, A., Navarro, J., Piazza, A.: Distributed storage and querying techniques for a semantic Web of scientific workflow provenance. In: IEEE International Conference on Services Computing (2010)
2. Nedim Alpdemir, M., Mukherjee, A., Paton, N.W., Fernandes, A.A.A., Watson, P., Glover, K., Greenhalgh, C., Oinn, T., Tipney, H.: Contextualised Workflow Execution in MyGrid. In: Sloot, P.M.A., Hoekstra, A.G., Priol, T., Reinefeld, A., Bubak, M. (eds.) EGC 2005. LNCS, vol. 3470, pp. 444–453. Springer, Heidelberg (2005)
3. Altschul, S.F., Madden, T.L., Schaffer, A.A., Zhang, J., Zhang, Z., Miller, W., Lipman, D.J.: Gapped BLAST and PSI-BLAST: A new generation of protein database search programs. *Nucleic Acids Research* 25 (1997)
4. Apache Web Server (Version 2.2.22), <http://httpd.apache.org/>
5. BaBar, <http://www-public.slac.stanford.edu/babar/>
6. Bhagwat, D., Chiticariu, L., Tan, W.-C., Vijayvargiya, G.: An annotation management system for relational databases. In: 30th ACM International Conference on Very Large Data Bases (2004)
7. Bose, R., Frew, J.: Lineage retrieval for scientific data processing: A survey. *ACM Computing Surveys* 37(1) (2005)
8. Callahan, S., Freire, J., Santos, E., Scheidegger, C., Silva, C., Vo, H.: VisTrails: Visualization meets data management. In: ACM SIGMOD International Conference on Management of Data (2006)
9. Chang, F., Dean, J., Ghemawat, S., Hsieh, W., Wallach, D., Burrows, M., Chandra, T., Fikes, A., Gruber, R.: BigTable: A distributed storage system for structured data. 7th USENIX Symposium on Operating Systems Design and Implementation (2006)
10. Event Tracing for Windows, <http://msdn.microsoft.com/en-us/library/bb968803.aspx>
11. Foster, I.T., Vckler, J.-S., Wilde, M., Zhao, Y.: A virtual data system for representing, querying, and automating data derivation. In: Scientific and Statistical Database Management Conference (2002)
12. Frew, J., Bose, R.: Earth System Science Workbench: A data management infrastructure for earth science products. In: Scientific and Statistical Database Management Conference (2001)
13. Frew, J., Metzger, D., Slaughter, P.: Automatic capture and reconstruction of computational provenance. *Concurrency and Computation* 20(5) (2008)
14. Filesystem in Userspace, <http://fuse.sourceforge.net>
15. Gehani, A., Lindqvist, U.: Bonsai: Balanced lineage authentication. In: 23rd Annual Computer Security Applications Conference. IEEE Computer Society (2007)
16. Gehani, A., Kim, M., Zhang, J.: Steps toward managing lineage metadata in Grid clusters. In: 1st Workshop on the Theory and Practice of Provenance (2009)
17. Gehani, A., Kim, M., Malik, T.: Efficient querying of distributed provenance stores. In: 8th ACM Workshop on the Challenges of Large Applications in Distributed Environments (2010)
18. Gehani, A., Kim, M.: Mendel: Efficiently verifying the lineage of data modified in multiple trust domains. In: 19th ACM International Symposium on High Performance Distributed Computing (2010)
19. Gehani, A., Tariq, D., Baig, B., Malik, T.: Policy-based integration of provenance metadata. In: 12th IEEE International Symposium on Policies for Distributed Systems and Networks (2011)

20. Glavic, B., Alonso, G.: Perm: Processing provenance and data on the same data model through query rewriting. In: 25th International Conference on Data Engineering (2009)
21. Graphviz, <http://www.graphviz.org/>
22. Green, T., Karvounarakis, G., Tannen, V.: Provenance semirings. In: 26th ACM Symposium on Principles of Database Systems (2007)
23. Groth, P., Moreau, L.: Representing distributed systems using the Open Provenance Model. *Future Generation Computer Systems* 27(6) (2011)
24. Heydon, A., Levin, R., Mann, T., Yu, Y.: The Vesta Approach to Software Configuration Management. Technical Report 168, Compaq Systems Research Center (2001)
25. Heinis, T., Alonso, G.: Efficient lineage tracking for scientific workflows. In: ACM SIGMOD International Conference on Management of Data (2008)
26. Holland, D.A., Braun, U., Maclean, D., Muniswamy-Reddy, K., Seltzer, M.: Choosing a data model and query language for provenance. In: 2nd International Provenance and Annotation Workshop (2008)
27. H2, <http://www.h2database.com>
28. Installable File System, <http://msdn.microsoft.com/en-us/windows/hardware/gg463062.aspx>
29. Influenza Data, National Institutes of Health, <ftp://ftp.ncbi.nlm.nih.gov/genomes/INFLUENZA/influenza.faa>
30. Java Data Base Connectivity, <http://www.oracle.com/technetwork/java/overview-141217.html>
31. Java Native Interface, <http://java.sun.com/docs/books/jni/>
32. Kementsietsidis, A., Wang, M.: On the efficiency of provenance queries. In: 25th International Conference on Data Engineering (2009)
33. Linux Audit, <http://people.redhat.com/sgrubb/audit/>
34. LLVM, <http://llvm.org>
35. lsof, <ftp://lsof.itap.purdue.edu/pub/tools/unix/lsof>
36. Apache Lucene, http://lucene.apache.org/core/old_versioned_docs/versions/3_0_1/queryparsersyntax.html
37. MacFUSE, <http://code.google.com/p/macfuse/>
38. Macko, P., Seltzer, M.: A general-purpose provenance library. In: 4th USENIX Workshop on the Theory and Practice of Provenance (2012)
39. Malik, T., Gehani, A., Tariq, D., Zaffar, F.: Sketching Distributed Data Provenance. In : Liu, Q., Bai, Q., Giugni, S., Williamson, D., Taylor, J. (eds.) *Data Provenance and Data Management in eScience*. SCI, vol. 426, pp. 85–108. Springer, Heidelberg (2013)
40. MATLAB, <http://www.mathworks.com/products/matlab/>
41. Miles, S., Deelman, E., Groth, P., Vahi, K., Mehta, G., Moreau, L.: Connecting scientific data to scientific experiments with provenance. In: 3rd IEEE International Conference on e-Science and Grid Computing (2007)
42. Moreau, L., Clifford, B., Freire, J., Futrelle, J., Gil, Y., Groth, P., Kwasnikowska, N., Miles, S., Missier, P., Myers, J., Plale, B., Simmhan, Y., Stephan, E., Van den Bussche, J.: The Open Provenance Model core specification (v1.1). *Future Generation Computer Systems* (2010)
43. MySQL, <http://www.mysql.com/>
44. Neo4j, <http://neo4j.org/>
45. Novel Information Gathering and Harvesting Techniques for Intelligence in Global Autonomous Language Exploitation, <http://www.speech.sri.com/projects/GALE/>
46. OpenBSM, <http://www.trustedbsd.org/openbsm.html>
47. Open Provenance Model, <http://openprovenance.org/>

48. Pancerella, C., Hewson, J., Koegler, W., Leahy, D., Lee, M., Rahn, L., Yang, C., Myers, J.D., Didier, B., McCoy, R., Schuchardt, K., Stephan, E., Windus, T., Amin, K., Bittner, S., Lansing, C., Minkoff, M., Nijssure, S., van. Laszewski, G., Pinzon, R., Ruscic, B., Wagner, A., Wang, B., Pitz, W., Ho, Y.L., Montoya, D., Xu, L., Allison, T.C., Green Jr., W.H., Frenklach, M.: Metadata in the collaboratory for multi-scale chemical science. In: Dublin Core Conference (2003)
49. Process Monitor, Windows Sysinternals,
<http://technet.microsoft.com/en-us/sysinternals/bb896645.aspx>
50. Rajgarhia, A., Gehani, A.: Performance and extension of user space file systems. In: 25th ACM Symposium on Applied Computing (2010)
51. Muniswamy-Reddy, K.-K., Holland, D.A., Braun, U., Seltzer, M.: Provenance-aware storage systems. In: USENIX Annual Technical Conference (2006)
52. Muniswamy-Reddy, K.-K., Braun, U., Holland, D.A., Macko, P., Maclean, D., Margo, D., Seltzer, M., Smogor, R.: Layering in provenance systems. In: USENIX Annual Technical Conference (2009)
53. Muniswamy-Reddy, K.-K., Macko, P., Seltzer, M.: Making a Cloud provenance-aware. In: 1st USENIX Workshop on the Theory and Practice of Provenance (2009)
54. Muniswamy-Reddy, K.-K., Macko, P., Seltzer, M.: Provenance for the Cloud. In: 8th USENIX Conference on File and Storage Technologies (2010)
55. Lineage File System, <http://crypto.stanford.edu/~cao/lineage.html>
56. Scalable Authentication of Grid Data Provenance,
<http://www.nsf.gov/awardsearch/showAward.do?AwardNumber=0722068>
57. Silva, C.T., Freire, J., Callahan, S.: Provenance for visualizations: Reproducibility and beyond. *Computing in Science and Engineering* 9(5) (2007)
58. SLAC National Accelerator Laboratory, <http://www.slac.stanford.edu/>
59. Support for Provenance Auditing in Distributed Environments,
<http://spade.csl.sri.com/>
60. Szomszor, M., Moreau, L.: Recording and Reasoning over Data Provenance in Web and Grid Services. In: Meersman, R., Schmidt, D.C. (eds.) *CoopIS/DOA/ODBASE 2003*. LNCS, vol. 2888, pp. 603–620. Springer, Heidelberg (2003)
61. Tariq, D., Ali, M., Gehani, A.: Towards Automated Collection of Application-Level Data Provenance. In: 4th USENIX Workshop on the Theory and Practice of Provenance (2012)
62. Tupelo project, NCSA, <http://tupeloproject.ncsa.uiuc.edu/node/2>
63. Windows Driver Kit,
<http://msdn.microsoft.com/en-us/windows/hardware/gg487428.aspx>
64. WebDAV, <http://www.webdav.org/>
65. Widom, J.: Trio: A system for integrated management of data, accuracy and lineage. In: 2nd Conference on Innovative Data Systems Research (2005)
66. Windows Management Instrumentation, [http://msdn.microsoft.com/en-us/library/aa394582\(v=VS.85\).aspx](http://msdn.microsoft.com/en-us/library/aa394582(v=VS.85).aspx)
67. Zhao, J., Goble, C.A., Stevens, R., Bechhofer, S.: Semantically Linking and Browsing Provenance Logs for E-science. In: Bouzeghoub, M., Goble, C.A., Kashyap, V., Spaccapietra, S. (eds.) *ICSNW 2004*. LNCS, vol. 3226, pp. 158–176. Springer, Heidelberg (2004)
68. Zhou, W., Sherr, M., Tao, T., Li, X., Loo, B., Mao, Y.: Efficient querying and maintenance of network provenance at Internet-scale. In: *ACM SIGMOD International Conference on Management of Data* (2010)