Bonsai: Balanced Lineage Authentication

Ashish Gehani Ulf Lindqvist
SRI International
E-mail: {ashish.gehani,ulf.lindqvist}@sri.com *

Abstract

The provenance of a piece of data is of utility to a wide range of applications. Its availability can be drastically increased by automatically collecting lineage information during filesystem operations. However, when data is processed by multiple users in independent administrative domains, the resulting filesystem metadata can be trusted only if it has been cryptographically certified. This has three ramifications: it slows down filesystem operations, it requires more storage for metadata, and verification depends on attestations from remote nodes.

We show that current schemes do not scale in a distributed environment. In particular, as data is processed, the latency of filesystem operations will degrade exponentially. Further, the amount of storage needed for the lineage metadata will grow at a similar rate. Next, we examine a completely decentralized scheme that has fast filesystem operations with minimal storage overhead. We demonstrate that its verification operation will fail with an exponentially increasing likelihood as more nodes are unreachable (because of being powered off or disconnected from the network). Finally, we present a new scheme, Bonsai, where the verification failure is significantly reduced by tolerating a small increase in filesystem latency and storage overhead for certification compared to filesystems without lineage certification.

1. Introduction

The utility of knowing the provenance of a piece of data can be judged by the range of domain-specific applications that have been developed to track it. Geographic Information Systems (GIS) standards include lineage metadata to track datasets [8]. Materials scientists use it to establish the pedigree of data used for designing safety-critical components [29]. Biologists must document their experiments to allow others to reproduce them [16]. It has been posited that Grid infrastructure should provide automated, application-transparent provenance collection in a standardized format [7]. Taverna [2] is being developed as part of $^{my}Grid$ [22] to address the need. Several applications of provenance metadata relate to data security. These include ensuring the reliability, auditability, reproducibility, ownership, and accreditation of data [13]. Despite this, the authenticity of provenance metadata is not certified or verified.

Diverse universities, research laboratories, and corporations are employing Grid computing for a range of applications [12]. Large volumes of data are being created, disseminated, and processed in distributed systems that span multiple administrative domains. To maintain accountability while the data is transformed by multiple parties, a consumer must be able to check its lineage and deem it trustworthy. If the integrity is not ensured, the consequences can be significant since such data is not easily reproduced, because it is often the product of employing substantial computational resources for extended periods of time. For example, each piece of information that physicists use from Fermilab's Collider Detector is the output of a month of processing dozens of terabytes of raw data [36]. Similarly, the cost to analyze a single protein stored in the Protein Data Bank is $200,000 [33]. The cost of producing such data precludes its availability from an alternate source. If its authenticity is not tracked from the time of creation, fraudulent modifications may go undetected and disrupt experiments that use it.

Lineage metadata used for security applications differs from that gathered for other purposes in one critical respect. It must be *complete* since omissions may alter the security semantics, such as which set of principals were responsible for producing a piece of data. This precludes the utilization of application-specific mechanisms for abstracting or reducing the metadata.

In Section 4, we propose an architecture for gathering lineage metadata in distributed systems. Section 5 explains how we exploit the independence of nodes for certifying

*This work was produced in part with support from the Institute for Information Infrastructure Protection (I3P) research program. The I3P is managed by Dartmouth College, and supported under Award number 2003-TK-TX-0003 from the U.S. Department of Homeland Security, Science and Technology Directorate. Points of view in this document are those of the author(s) and do not necessarily represent the official position of the U.S. Department of Homeland Security, the Science and Technology Directorate, the I3P, or Dartmouth College.

and verifying lineage metadata. Section 6 describes how current application-specific lineage accumulates, the issues involved with using a completely decentralized implementation of our architecture, and finally a hybrid scheme, Bonsai, that trades metadata storage for verification reliability. We compare the alternatives in Section 7.

2. Motivation

To clarify the problem domain, we describe an example of a distributed computation for genome analysis. The GADU[1] system is designed to automate the assignment of functions to genes [28]. We trace the provenance of a single result in the final database, as illustrated in Figure 1. Periodically, a query is made to the NCBI[2] [23], JGI[3] [17], TIGR[4] [38], PDB[5] [25], and Swiss-Prot [37] databases. If any new data is found, it is downloaded to the GADU server. The Pegasus planner [9] dispatches sequence data to hundreds of remote nodes. At each node, reference data is drawn from BLAST [3], PFAM [5], BLOCKS [14], and THMM [19] databases for different types of comparative analyses. The result from each is then output to a database.

In Figure 1, data sources are depicted with rectangular boxes, while computational processes are drawn in boxes with round corners. The GADU server and database of final outputs are in the user's administrative domain. Hence, they are colored green to signify that they are completely trusted. The NCBI, JGI, TIGR, PDB, Swiss-Prot, BLAST, PFAM, BLOCKS, and THMM databases are colored yellow to indicate that they are trusted despited being in different administrative domains from that of the user. Finally, the computation nodes are marked in red to represent the fact that no prior trust relationship exists with these nodes. In this case, these are Grid nodes at research institutions. They could instead be personal computers in people's homes, as is the case with projects like SETI [4]. It is therefore necessary to audit the nodes' claims to maintain their accountability. We do this for lineage information by forcing them to cryptographically commit to the metadata they generate. Without our proposed assurance, any node in an external administrative domain could alter the lineage of the results fed into the database of final outputs without being traceable.

3. Goal

Our goal is to reliably be able to determine the lineage of a piece of data. Therefore, the metadata should have the following properties. First, the provenance must be *authentic*. A principal must not be able to create, append, or modify an element as another principal. Subsequently, other principals must be able to validate the element. Second, only operations and inputs *necessary* for reconstructing the data should be noted in the lineage metadata. For example, if a sequence of idempotent operations is performed on the data, only one should be recorded. Third, the lineage must be *complete*. It must enumerate all the inputs used to construct the data object in question. If an object's metadata satisfies these properties, its lineage can be accurately characterized.

[1]Genome Analysis and Database Update
[2]National Center for Biotechnology Information
[3]Joint Genome Institute
[4]The Institute for Genomic Research
[5]Protein Data Bank

4. Architecture

We now explain in detail what constitutes the lineage of a piece of data. The granularity at which we track the provenance of an object affects the overhead introduced. If we attempt to trace and record the details of every operation connected to the object, the system's performance will perceptibly degrade and the metadata will grow to need more space than the data. Instead, we exploit the fact that files cross administrative boundaries as integral units. Thus, when analyzing the trustworthiness of an object, analysis at file granularity suffices.

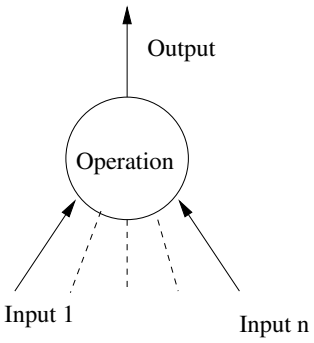

Figure 2. A *primitive operation* transforms a set of input files into a single output file.

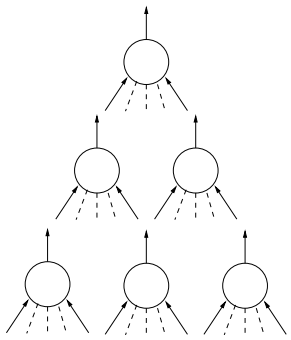

Figure 3. An object's lineage is a collection of primitive operations assembled into a *compound operation* tree.

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 to a file, a primitive operation has been performed. We denote the primitive operation of Figure 2 as $(O, E, I_1, \ldots, I_n)$, where O is the output of the operation executed by E using inputs $I_1, \ldots, I_n$. If a pro-

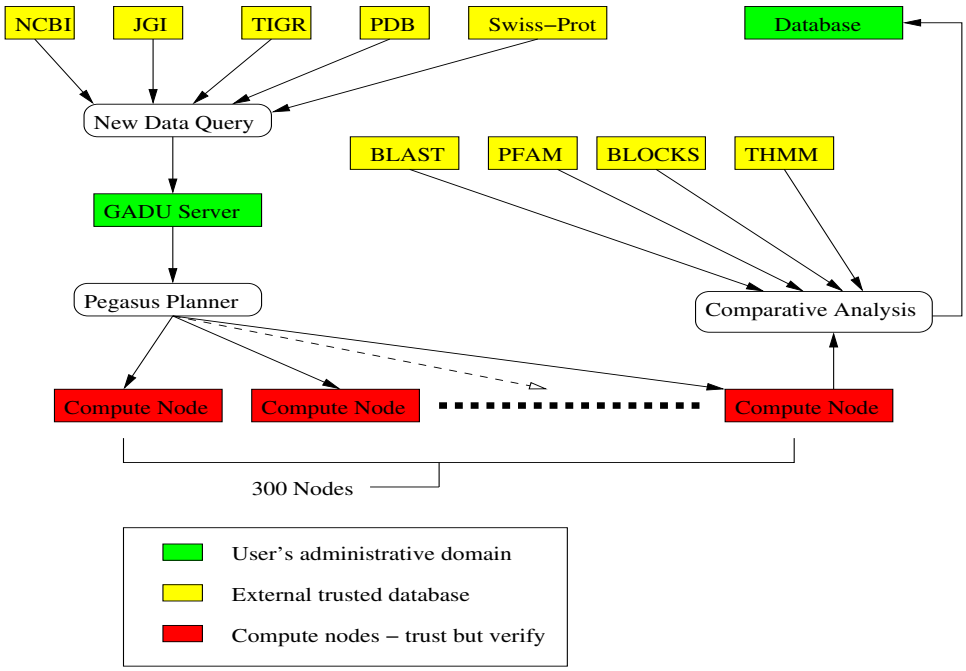

Figure 1. Distributed operations combine information from multiple data sources. The final output's relationship to the original input is not self-evident. Since intermediate nodes are only partially trusted, their claims must be verified.

cess writes out to a number of files, a separate instance of the representation in Figure 2 is used for each output file. Primitive operations are combined into a *compound operation*, as depicted in Figure 3. Each vertex represents the execution of a different process. For instance, if the result of appending together several datasets (by a program such as Linux *cat*) is then sorted into a particular order (using another program, such as Linux *sort*, that executes as a separate process), then the combination of appending and sorting is a compound operation. Thus, every data object is the result of a compound operation that can be represented by a lineage tree. The set of all primitive operations in the tree serves as the abstract description of the lineage.

We do not store the details of the process in our representation of a primitive operation. Instead we note the identity of the user who executed the process. We do this since the end user is interested in who modified the data en route. This identity must have global semantics. We assume the availability of a public key infrastructure [21]. However, any distributed mechanism for resolving identities, such as linked local namespaces [1] or a web of trust [27], can be used instead. A comprehensive provenance record would include further detail of the specific operation performed, such as the binary executed, libraries called, environment variable values, and hardware configuration. Other research projects focus on the collection of such information [26]. This is complementary to our work since their record of the invocation of a program can be included in our representation of a primitive operation. Our lineage certification algorithms would continue to operate without any alterations.

At first glance, our definition of a compound operation may appear to introduce false dependencies. One may expect to be able to provide a more precise dependency set by using threads, system calls, or assembly instructions as the definition of an operation. For example, one could construct the system call control flow graph of a process and trace the possible execution paths to an output. From an information flow standpoint, only the inputs that feed into the output should be part of its dependency set. However, our goal is tracking the lineage of the final output of a compound operation. An execution sequence in a process, P, that does not affect the primitive operation's output but does affect the input of another process, $\hat{P}$, is called a *side effect*. An input, I, that produces only a side effect would not be included in the dependency set calculated based on information flow to the output. However, the output of $\hat{P}$ may then be utilized as part of the compound operation. As a result, the input I would not be included in the lineage even though it should have been. This necessitates the conservative approach that we follow.

5. Design

Researchers from the Globus and Condor projects recently argued that the lack of transparent file access and the inability to use unmodified programs has hindered the adoption of Grid computing [18]. They are addressing the issue by building interposition agents to provide this facility. Our work is complementary to this and based on the same assumptions. Current Grid provenance collection focuses on capturing application-specific workflows [35].

Our approach for recording provenance in a distributed setting is guided by the following insights:

- The absence of a globally trusted computing base on each node would appear to preclude any assurances about its output. However, we can still guarantee properties of data passing through it. The structure of the problem allows us to avoid relying solely on a node's correct operation. Below we assume that each node is in a different administrative domain. The same argument holds for groups of nodes from different domains.

  Every edge in the lineage graph has two ends. If they are incident on two different nodes, the output generated on one node serves as the input on another node. If a node makes a false claim about its input, there will be a discrepancy with its predecessor's output. If a node makes a false claim about its output, there will be a discrepancy with its successor's input. This property allows us to detect fraudulent claims if even a single node in the lineage graph is operating correctly (since the discrepancy will be detected by the transitive closure of the verification operation).

- A node may be operating under an arbitrary protection domain. It is therefore not possible to force it to perform specific actions. However, we can require it to commit claims about operations it performs, failing which data originating from the node will be rejected. Each time a node generates an output, it must provide the output with provenance metadata. This includes the provenance of each input and a signed hash of which inputs and output file were involved. Once the output has been provided to another node, the claim is considered committed.

By introducing the lineage collection and certification at the operating system level (in the file system) we aim to produce a solution that is transparent to existent applications, thereby easing deployment. We now describe the overall architecture of our infrastructure, leaving the description of the alternative protocols to Section 6.

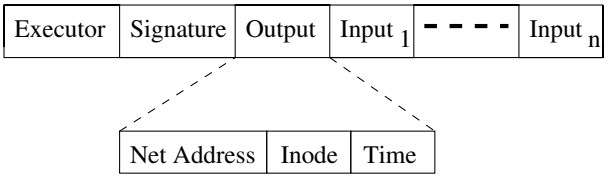

Figure 4. A primitive operation is stored in this format. Each input is the output from a previous operation (and hence has the same format as the output shown).

5.1. Representation of a Primitive Operation

If all the input data, application code, and system libraries used to generate an output are available, an operation can be verified by repeating it. In practice, programs often read data from ephemeral sources such as network sockets, filesystem pipes, or devices that provide sources of randomness. This prevents the program's output from being independently validated in a distributed environment since the verifier must trust the original executor's claims about the contents derived from these sources. Since the executor has the freedom to alter the ephemeral inputs to yield a preferred output, checking the operation by repeating it does not increase the likelihood that the claimed operation was the one that was previously performed. Hence, our checks are restricted to the persistent files read and written by a process.

The format of a primitive operation is depicted in Figure 4. The first element, E, identifies the principal that created the metadata. We use a three-part field for E. The first is for a user identifier (that is specific to the local host), the second for a public network address, and the third for an internal intranet address. (All the users behind a NAT'ed (Network Address Translated) firewall are associated with a single public IP address.) The second element, S, is the output of a digital signature that commits the executor E to the output file, O, and the set of inputs $I_1, \ldots, I_n$ used in its creation. We use a signature $S = \textsc{Sign}_{K_E}(E, O, I_1, \ldots, I_n)$, where K_E is the principal E's private signing key. Sufficient space is allowed for a cryptographically strong digital signature. (160 bits is currently considered safe [6].) If there are no inputs (as occurs when an object has been captured directly from a sensor), then the signature's parameter is just the output file, O. Such leaf nodes may utilize weaker signatures, in which case the field is padded.

Each input and output file is represented with a globally unique identifier. We use a four-part field to represent an input or output file. The first two parts are the external IP and internal intranet addresses of the host where the operation occurred. The third is the *inode* (or equivalent filesystem

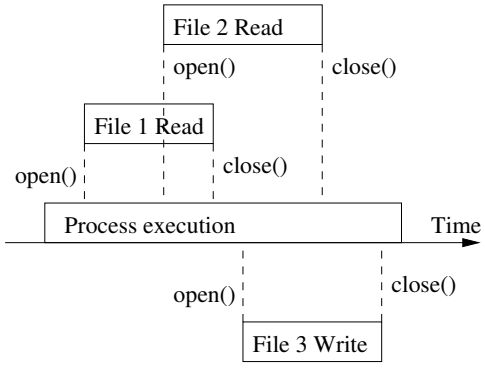

Figure 5. When a file is closed after being written, its provenance includes every file that was read by the process. The provenance of *File 3* is the list {*File 1, File 2*}.

identifier) of the file containing the input or output. The fourth is the time when the input or output file was read or modified, respectively. The time is intended to disambiguate different versions of the same file to avoid cycles in the lineage graph.

5.2. Lineage Certification

We automate the procedure as follows. When the system boots, a *lineage daemon* is initialized. This maintains a table T_{PID} that maps process identifiers to live provenance metadata. Each process's entry in T_{PID} contains an `accessed` list of all files that have been read by it, and a `modified` list of all files that have been written by it.

Our code intercedes on file *open(), close(), read(), write()* system calls. [6] When a file *open()* call occurs, a check is done to see if the calling process has an entry in T_{PID}. If not, an entry is created and populated with empty `accessed` and `modified` lists. When a *read()* operation occurs, the file being read is added to the calling process's `accessed` list. Similarly, when a *write()* occurs the file is added to the writing process's `modified` list.

When a *close()* occurs, the calling process's `modified` list is checked. If the file has actually been written to (as opposed to just being opened for writing or just having been read from), the `modified` list will contain it. In this case, the *accessed* list of the process is retrieved from T_{PID}. It contains the list of files $\{I_1, \ldots, I_n\}$ that have been read during the execution of the process up to the point that the output file O was closed. This is illustrated in Figure 5. A certified primitive operation is constructed by appending the process owner's identity E, a signature S binding the output to the inputs utilized, the output O, and the inputs $\{I_1, \ldots, I_n\}$, where $S = Sign(E, O, I_1, \ldots, I_n)$. The representation shown in Figure 4 is used for the inputs and output, so that files with the same name on distinct hosts, or with different contents at other points in time on the same host, are clearly disambiguated.

In practice, the above step occurs after all references to the file become inactive. This possibility arises since multiple valid concurrent references may result after a single *open* call. Such a situation occurs when a process spawns multiple threads and passes them a file descriptor. Equivalently, this occurs when a process makes a *fork()* call, creating another process that has copies of all its active file descriptors. Alternatively, this can occur if a part of the file was mapped to memory. Once all active file descriptors are closed and relevant memory blocks are unmapped, it is safe to certify the lineage of the output file.

[6] Though these calls are specific to Unix-like environments, their analogues exist in other filesystems, notably that of Windows NT.)

Algorithm 5.1: CHECKLINEAGE(D)

$\{E, S, O, I_1, \ldots, I_n\} \leftarrow$ GETROOT(D)
OUTPUT(E)
$P_E \leftarrow$ PKILOOKUP(E)
if $I_1, \ldots, I_n = \{\}$
  then
    $Result \leftarrow$ VERIFY(P_E, S, O)
    if $Result =$ FALSE
      then CheckFailed
  else
    $Result \leftarrow$ VERIFY($P_E, S, O|I_1|\ldots|I_n$)
    if $Result =$ TRUE
      then
        for $i \leftarrow 1$ **to** n
          do CHECKLINEAGE(I_i)
      else CheckFailed

5.3. Lineage Verification

As the provenance of a piece of data increases, verifying its lineage rapidly becomes a nontrivial operation. Therefore, lineage is verified programmatically and only on demand. Algorithm 5.1 shows how this is done by recursively checking each element. The GETROOT function finds the vertex whose output matches its input parameter, D. Section 6 describes three alternative protocols for lineage certification. Each uses a significantly different implementation of the GETROOT function during verification. In each case, GETROOT yields the same output. The first element is the identity E of the owner of the process that resulted in the primitive operation defined by the vertex in question. The function PKILOOKUP maps the identity to the user's public key, P_E, which is needed to verify signatures they have generated. As the CHECKLINEAGE function recurses, the output from this step enumerates the identities that have

modified any of the data utilized in producing the object. The user's signature validating the set of inputs used to produce the output is verified (even if there are no inputs) with the VERIFY operation. (In the algorithm below, | refers to catenation of data.) Then the lineage of each input (if any exist) is recursively checked. If at any point a signature check fails, the function halts and issues an alert.

6. Protocols

We now describe three alternative ways of certifying and verifying lineage. Section 7 evaluates each scheme's impact on system performance. All three assume a central source of trust (which can be offline) that provides each user with a cryptographic identity.

Distributed systems have provided data integrity certification and verification for several decades. However, the attestation provides assurance only about the current state of the data, not its history. Without protocols of the type described below, a principal making modifications early in the history of a piece of data could claim not to have done so. It is this *non-repudiability* property that distinguishes lineage certification protocols from earlier data authentication schemes.

6.1. Cumulative Lineage

The first protocol we consider is one where lineage information is accumulated along a workflow. This mechanism is used to prevent repudiation in Grid computing environments [32]. When an operation is performed, the signed lineage of each input is extracted and added to the metadata of each output. The resulting metadata is hashed and signed. Since all the lineage information can be retrieved from an object's metadata, no network connectivity is required for verification. However, the space required for the metadata grows exponentially in the number of steps used to process the data (assuming multiple input files are used to produce a single output). Over time, the size of lineage metadata will dominate the size of the actual data. During verification, Algorithm 5.1's GETROOT function can extract each vertex from the object's metadata.

6.2. Decentralized Lineage

Cryptographers noticed the exponential increase in space required to store the signatures needed to verify that a document was created by a particular workflow. This led to the development of *aggregate signatures* that allow each document to be accompanied by a single fixed size signature [41]. However, verification requires the entire tree of signatures for documents that are part of the lineage (and are unlikely to be present at the node where verification is occurring). This is effectively a decentralized protocol, where

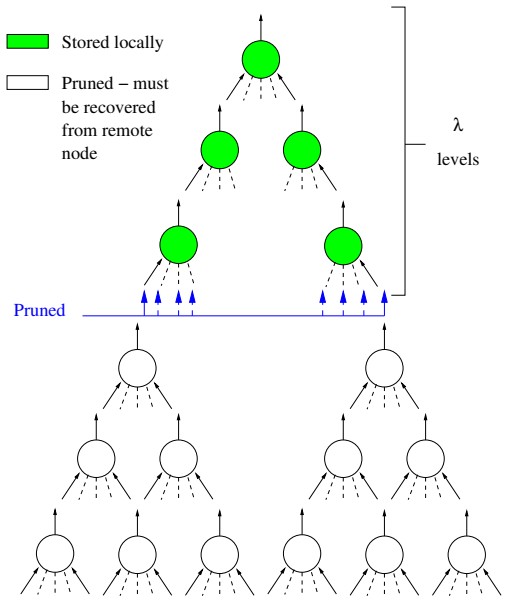

Figure 6. Bonsai *prunes* out lineage graph information from vertexes generated more than λ steps earlier. It replaces each vertex at level $(\lambda + 1)$ with a pointer to the node where that vertex was generated originally. During verification this pointer is used to reconstruct the subtree rooted at the pruned vertex.

the representation of each primitive operation is stored locally at the node where the output file was created or modified.

The decentralized protocol does address the issue of exponential growth in metadata size. It leverages the high level of redundancy in the lineage metadata of succeeding generations of data. Since the details of a primitive operation are stored only at the node where it occurs, the metadata for an object then includes pointers to the node from where the details can be retrieved, along with a cryptographic witness that ensures that the details cannot be altered in the interim. A *lineage daemon* must run on each node and be able to service a query about such a pointer, returning the entire representation of the corresponding primitive operation. The redundancy in the metadata is eliminated, resulting in constant storage overhead.

During lineage verification, the GETROOT function of Algorithm 5.1 must resolve each input's pointer by contacting the appropriate lineage daemon. We see that reduction in storage overhead for certification translates into a corresponding increase in temporal overhead for verification. The certification storage overhead has a commensurate effect on filesystem latency when *open()* and *close()* operations occur. In principle, the tradeoff would be acceptable

Steps / Workload	1	2	3	4	5
Instruction	0.4 KB	3 KB	31 KB	253 KB	2 MB
Research	0.2 KB	0.8 KB	2 KB	8 KB	29 KB
Web	1 KB	39 KB	1 MB	29 MB	813 MB
Windows	0.2 KB	0.8 KB	2 KB	9 KB	30 KB

Table 1. The space needed to represent the *cumulative lineage* of an object depends on two factors - the average number of files used to produce each intermediate output, and the number of times data is repeatedly processed.

since certification is on the critical path of filesystem operations while verification occurs only on demand when a user interrogates the system to determine the lineage of a file. In practice, the dependence on the network for verification has a significant side effect. It requires every node responsible for a vertex in an object's lineage graph to be accessible online at the time of verification. If even one node is offline, the verification cannot complete.

6.3. Bonsai: A Hybrid Scheme

To reduce the likelihood of being unable to retrieve the details of a lineage vertex, we could replicate the information at several other nodes. The probability of all relevant nodes being unreachable at the same time would drop rapidly as the number of replicas increases. This would then introduce new overhead for selecting the nodes at which to replicate each piece of metadata. In particular, since the nodes from which data originated would also need to maintain copies of the vertex, the replica could be propagated back along a lineage graph. However, this would still require network traffic to be generated every time a local filesystem *close()* operation completed. Instead, we use a more efficient variant of the idea.

Bonsai is a hybrid scheme. It maintains the entire lineage of a piece of data, similarly to how the *cumulative lineage* scheme would. However, it differs in one respect. When a lineage tree grows past a predefined threshold number of levels λ, it *prunes* the lineage graph, as illustrated in Figure 6. When it prunes the lineage graph, it leaves a pointer at each vertex where the subtree has been removed. This pointer is managed in exactly the same way as the *decentralized lineage* scheme. In fact, the decentralized lineage scheme can be viewed as a special case of Bonsai, with $\lambda = 1$. (The protocol is called Bonsai since it prunes the local lineage tree of an object to ensure that it never grows too large.)

As λ increases, the lineage metadata requires more storage. However, it also results in more nodes containing copies of each vertex since the metadata is transparently copied with the data up to λ times. The availability of other copies decreases the probability that a vertex cannot be retrieved during verification. This is achieved without introducing a new protocol to replicate the lineage metadata. In a system where most nodes are reachable, if the lineage graph is δ levels deep, the entire graph of an object can be verified by contacting nodes at every $\frac{\delta}{\lambda}$ levels, yielding a large speedup in verification. Thus, using Bonsai with a suitable level of λ allows the overhead of lineage certification to be traded for better performance of lineage verification.

7. Evaluation

To understand the benefits of using Bonsai, we modeled the effect of several workloads on each of the three protocols described in Section 6 if they were to be deployed using our architecture for automated lineage certification (as described in Section 5).

7.1. Workload

The choice of where to store the lineage metadata results in different implementations of our architecture. Each scheme has a different impact on the latency of filesystem operations, the storage overhead at a node, the cumulative storage used by the system, the time to verify the lineage of a piece of data, and the likelihood that a verification operation can be completed successfully when some fraction of the network nodes are unreachable. To compare the schemes, we modeled their performance using statistics from four sets of filesystem traces [30].

The first set is derived from twenty workstations in a laboratory for teaching undergraduate classes. The second set of traces is from thirteen desktop computers of a research group's graduate students, faculty, and administrative staff. The third set of traces is from a host running a Web server and Postgres database. The fourth set is from eight desktop computers running Windows. The four traces are labeled *Instruction*, *Research*, *Web*, and *Windows*, respectively. Close to one month of filesystem activity is used to compute the statistics of each set of traces. Roselli's dissertation [31] provides further detail.

Workload \ Steps	1	2	3	4
Instruction	0.04	0.05	0.11	1.72
Research	0.05	0.05	0.04	0.04
Web	0.06	0.13	6.42	997.5
Windows	0.07	0.04	0.04	0.04

Table 2. The latency introduced (in ms) into a filesystem *open()* call to read the *cumulative lineage* of an object depends on the amount of metadata that needs to be read.

Workload \ Steps	1	2	3	4
Instruction	0.20	0.28	0.32	0.84
Research	0.16	0.19	2.39	3.1
Web	0.16	0.24	4.82	579.14
Windows	0.16	0.50	5.34	3.17

Table 3. The latency added (in ms) to a *close()* call when writing out lineage.

7.2. Certification Overhead

We estimate the average amount of storage needed for a single file's lineage metadata as follows. The number of *read()* and *write()* operations performed per process is calculated for each workload. Based on the number of distinct file *open()* and *close()* operations performed, we calculate how many of those correspond to distinct files. We can compute this since less than 0.2% of the runs include both *read()* and *write()* calls. (A *run* is the sequence of filesystem calls made by a process between calling *open()* and *close()* on a file.) This allows us to estimate how many inputs are utilized for each file that has been written to by a process. Since the file access patterns are stable over varying periods of time [31], we normalize the trace activity over the period of recording. This is then used to estimate the growth of the lineage metadata as the data is repeatedly processed. The results are shown in Table 1.

In the case of the *cumulative lineage* protocol, this is the average amount of lineage metadata that would be stored with a file. The rapid growth in the metadata is visible in all workloads. However, in the case of the *Web* workload, which consists of a single process that executes for a long period of time, all the files read by the server become part of the lineage of any file that is modified by it. This introduces a very large amount of metadata in the output files. Clearly, this protocol is untenable for such workloads. The *Research* and *Windows* workloads reflect common desktop usage patterns. They incur relatively low storage overhead costs for maintaining the entire lineage tree with an object. We can infer from this that interactive use of the computer involves far fewer reads per file written out. The *Instruction* workload's overhead is less than that of the *Web* workload, but it is still high enough that the *cumulative lineage* protocol is untenable.

Table 2 shows the latency that is introduced into a filesystem *open()* call to read the lineage metadata. Similarly, Table 3 shows the time taken to write out the metadata during a *close()* call. The data was gathered on Mac OS 10.4.9 running on a 2 GHz Intel Core Duo. Each table entry is the average of 100 measurements. In both tables, the delay is seen to grow rapidly. However, the overhead is not noticeable relative to the time taken for the filesystem call, when the lineage graph is of moderate size. For example, the *Instruction* workload adds only 1.72 ms to the *open()* system call even when it is reading a lineage graph with four levels. This fact forms the basis for why Bonsai can provide fast lineage certification. If Bonsai was utilized with $\lambda = 3$, then the maximum latency introduced into the *open()* call would be 6.42 ms (which occurs in the case of the *Web* trace) for any of the workloads. Yet with $\lambda = 3$, we get a substantial improvement in verification reliability as can be seen from Figure 7.

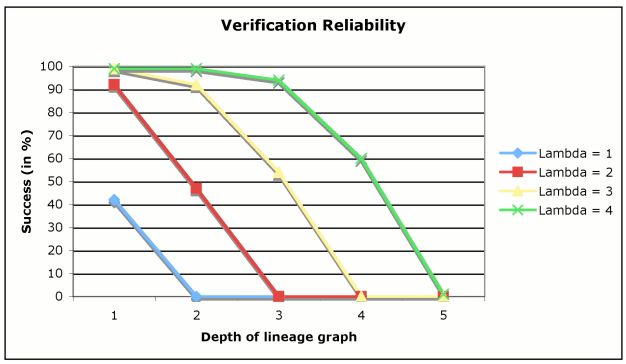

Figure 7. As λ is increased, the verification reliability increases significantly because there are more copies of the lineage information, allowing the verification operation to complete even if the original node is unreachable.

7.3. Verification Reliability

When Bonsai is used with $\lambda = 1$, it provides equivalent functionality to the *decentralized lineage* protocol. From Figure 7, we can see that the reliability of lineage verification drops rapidly to 0 as the lineage graph's size increases. The data in Figure 7 is for the case when each individual node in the distributed system may be unreachable with

probability 0.1. A node may be unreachable because it has been powered off, has been disconnected from the network (as could occur if it was portable), or due to a network partition (as could occur if ad hoc networking were used). In the case of $\lambda = 1$, no other node holds a copy of the lineage operations certified at any node. Therefore, when a node is offline, if it is part of the lineage being verified, the operation will fail. As Bonsai uses a larger value of λ, the verification operation can recover the information from up to λ other nodes en route between the origination point and the current location of the data. This drastically reduces the likelihood of failure.

8. Related Work

Data provenance has a range of applications. HP SRC's Vesta [15] uses it to make software builds incremental and repeatable. Lineage File System [34] records the input files, command line options, and output files when a program is executed. Its records are stored in an SQL database that can be queried to reconstruct the lineage of a file. Provenance-Aware Storage System [26] augments this with details of the software and hardware environment. Since these systems were designed for use in single administrative domains, the metadata is not certified. Further, the provenance is stored separately from the objects. Consequently, when data files are moved between systems, the provenance may be lost. In contrast, our scheme ensures provenance and its certification is transparently transferred with the data.

Several distributed systems have been built to help scientists track their data. Chimera [10] 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. $^{my}Grid$ [42] allows users to model their workflows in a Grid environment. It is designed to aid biologists in performing computer-based experiments. CMCS [24] is a toolkit for chemists to manage experimental data derived from fields like combustion research. It is built atop WebDAV [39], a Web server extension that allows clients to modify data on the server. ESSW [11] 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. Trio [40] uses a data warehouse. It uses the lineage of data to automatically compute its accuracy. Bose and Frew's survey [7] identifies a number of other projects that aid in retrieving the lineage of scientific data. These systems all use a client-server architecture, where a user's data resides only in that user's domain and that of the server. Thus, metadata integrity is ensured. In contrast, we address the case where data moves through numerous administrative domains while being processed. Further, traditional primitives for metadata manipulation do not scale for provenance-based applications [20]. Our schemes are specifically designed to address the issue.

9. Conclusion

Transferring lineage metadata along with the data that it is related to creates an exponentially increasing number of copies of it. This redundancy is particularly problematic since it can make lineage metadata so voluminous that it surpasses the space needed for actual data. The key to avoiding the problem is to leave lineage details at the nodes where the operations occur and forwarding cryptographic commitments to prevent repudiation. Subsequently, each node can be queried when lineage derived on it is being verified. However, this creates reliance on remote nodes for lineage reconstruction and verification. The Bonsai protocol forwards a small fraction of the lineage tree with the data, using a small amount of storage and introducing a small increase in filesystem *open()* and *close()* system calls. In exchange it provides high reliability during lineage verification operations. For example, with the workloads examined, forwarding a three-level tree introduces metadata storage overhead on the order of kilobytes and latency on the order of milliseconds, with an order of magnitude increase in verification reliability.

References

[1] Martin Abadi, On SDSI's linked local name spaces, 10th Computer Security Foundations Workshop, 1997.

[2] M. Nedim Alpdemir, Arijit Mukherjee, Norman W. Paton, Alvaro A. A. Fernandes, Paul Watson, Kevin Glover, Chris Greenhalgh, Tom Oinn, and Hannah Tipney, Contextualised workflow execution in myGrid, Proceedings of the European Grid Conference, Lecture Notes in Computer Science, Volume 3470, Springer-Verlag, 2005.

[3] S. F. Altschul, T. L. Madden, A. A. Schaffer, J. Zhang, Z. Zhang, W. Miller, and D. J. Lipman, Gapped BLAST and PSI-BLAST: A new generation of protein database search programs, Nucleic Acids Research, Vol. 25, 1997.

[4] D. Anderson, J. Cobb, E. Korpela, M. Lebofsky, and D. Werthimer, SETI@home: An experiment in public-resource computing, Communications of the ACM, Vol. 45(11), 2002.

[5] A. Bateman, E. Birney, L. Cerruti, R. Durbin, L. Etwiller, S. R. Eddy, S. Griffiths Jones, K. L. Howe, M. Marshall, and E. L. Sonnhammer, The Pfam protein families database, Nucleic Acids Research, Vol. 30, 2002.

[6] Dan Boneh, Ben Lynn, and Hovav Shacham, Short signatures from the Weil pairing, Proceedings of Asiacrypt, Lecture Notes in Computer Science, Vol. 2248, 2001.

[7] R. Bose and J. Frew, Lineage retrieval for scientific data processing: A survey, ACM Computing Surveys, Volume 37(1), 2005.

[8] D. G. Clarke and D. M. Clark, Lineage, Elements of Spatial Data Quality, 1995.

[9] E. Deelman, J. Blythe, Y. Gil, C. Kesselman, G. Mehta, K. Vahi, K. Blackburn, A. Lazzarini, A. Arbree, R. Cavanaugh, and S. Koranda, Mapping abstract complex workflows onto Grid environments, Journal of Grid Computing, Vol. 1(1), 2003.

[10] I. T. Foster, J. S. Vockler, M. Wilde, and Y. Zhao, Chimera: A virtual data system for representing, querying, and automating data derivation, SSDBM, 2002.

[11] J. Frew and R. Bose, Earth System Science Workbench: A data management infrastructure for earth science products, SSDBM, 2001.

[12] Globus, `http://www.globus.org/alliance/projects.php`

[13] C. Goble, Position statement: Musings on provenance, workflow and (semantic web) annotations for Bioinformatics, Workshop on Data Derivation and Provenance, Chicago, 2002.

[14] S. Henikoff, J. G. Henikoff, and S. Pietrokovski, Blocks+: A non-redundant database of protein alignment blocks derived from multiple compilations, Bioinformatics, Vol. 15, 1999.

[15] A. Heydon, R. Levin, T. Mann, and Y. Yu, The Vesta approach to software configuration management, Technical Report 168, Compaq Systems Research Center, 2001.

[16] H. V. Jagadish and F. Olken, Database management for life sciences research, SIGMOD Record, Vol. 33, 2004.

[17] Department of Energy Joint Genome Institute, `http://www.jgi.doe.gov/`

[18] Sander Klous, Jamie Frey, Se-Chang Son, Douglas Thain, Alain Roy, Miron Livny, and Jo van den Brand, Transparent access to Grid resources for user software, concurrency and computation: Practice and experience, Vol. 18(7), 2006.

[19] Anders Krogh, Prediction of transmembrane helices in proteins, `http://www.cbs.dtu.dk/services/TMHMM/`

[20] David Liu, Michael Franklin, Jim Garlick, Ghaleb Abdulla, and Marcus Miller, Scaling up data provenance and smart recomputation for scientific workflows, Poster at International Conference for High Performance Computing and Communications, 2005.

[21] U. Maurer, Modelling a Public-Key Infrastructure, ESORICS '96, Lecture Notes in Computer Science, Vol. 1146, Springer-Verlag, 1996.

[22] $^{my}Grid$, `http://www.mygrid.org.uk`

[23] National Center for Biotechnology Information, `http://www.ncbi.nlm.nih.gov/`

[24] C. Pancerella, J. Hewson, W. Koegler, D. Leahy, M. Lee, L. Rahn, C. Yang, J. D. Myers, B. Didier, R. McCoy, K. Schuchardt, E. Stephan, T. Windus, K. Amin, S. Bittner, C. Lansing, M. Minkoff, S. Nijsure, G. v. Laszewski, R. Pinzon, B. Ruscic, Al Wagner, B. Wang, W. Pitz, Y. L. Ho, D. Montoya, L. Xu, T. C. Allison, W. H. Green, Jr., and M. Frenklach, Metadata in the collaboratory for multi-scale chemical science, Dublin Core Conference, 2003.

[25] Protein Data Bank, `http://www.rcsb.org/pdb/`

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

[27] M.K. Reiter and S.G. Stubblebine, Toward acceptable metrics of authentication, 18th IEEE Symposium on Security and Privacy, 1997.

[28] Alex Rodriguez, Dinanath Sulakhe, Elizabeth Marland, Veronika Nefedova, Michael Wilde, and Natalia Maltsev, Grid enabled server for high-throughput analysis of genomes, Workshop on Case Studies on Grid Applications, 2004.

[29] J. L. Romeu, Data quality and pedigree, Material Ease, 1999.

[30] D. Roselli, J. R. Lorch, and T. E. Anderson, A comparison of file system workloads, Proceedings of the Annual USENIX Technical Conference, 2000.

[31] Drew Roselli, Long-term File System Characterization, Ph.D. Thesis, Department of Computer Science, University of California at Berkeley, 2001.

[32] P. Ruth, D. Xu, B. K. Bhargava, and F. Regnier, E-notebook middleware for acccountability and reputation based trust in distributed data sharing communities, Proceedings of the 2nd International Conference on Trust Management, Springer-Verlag Lecture Notes in Computer Science, Vol. 2995, 2004.

[33] Andrej Sali, 100,000 protein structures for the biologist, Nature Structural Biology, Volume 5, 1998.

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

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

[36] Stefan Stonjek, Morag Burgon-Lyon, Richard St. Denis, Valeria Bartsch, Todd Huffman, Elliot Lipeles, and Frank Wurthwein, Using SAM datahandling in processing large data volumes, UK e-Science All Hands Meeting, 2004.

[37] Swiss-Prot Protein Knowledgebase, `http://us.expasy.org/sprot/`

[38] The Institute for Genomic Research, `http://www.tigr.org/`

[39] `http://www.webdav.org/`

[40] J. Widom, Trio: A system for integrated management of data, accuracy and lineage, Conference on Innovative Data Systems Research, 2005.

[41] Yongdong Wu, Efficient authentication of electronic document workflow, Conference on Information Security and Cryptology, Springer-Verlag Lecture Notes in Computer Science, Vol. 3822, 2005.

[42] J. Zhao, C. A. Goble, R. Stevens, and S. Bechhofer, Semantically linking and browsing provenance logs for E-science, ICSNW, 2004.