Provenance-*Only* Integration

Ashish Gehani Dawood Tariq
SRI International
{ashish.gehani,dawood.tariq}@sri.com

Abstract

As provenance records are collected from an increasingly diverse set of sources, the need to integrate them grows. The alternative approach of reconciling semantics scales when the records are queried infrequently. However, as the use of provenance grows, normalizing the diverse provenance via formal integration will yield better query performance. We describe two motivating cases for integrating provenance *only*, provide an initial formal model for integration that is domain-agnostic, and identify a possible direction for optimizing the integration process itself.

1. Introduction

The provenance of data can either be explicitly collected or retrospectively reconstructed. By interposing in systems that transform data, corresponding provenance metadata can be recorded. Alternatively, post-fact analysis of artifacts can be performed to infer the provenance relationships between pieces of data. Finally, provenance may be constructed by a hybrid approach that combines instrumentation and inference.

Regardless of the methodology by which provenance is obtained, the information is increasingly likely to arrive from a heterogeneous set of sources (rather than a single source or homogeneous group). The heterogeneity here refers to the variation in the identifiers, syntax, ontologies, completeness, and fidelity of the provenance elements.

TaPP '14, June 12–13, 2014, Cologne, Germany.

http://dx.doi.org/10.1145/

When a diverse set of provenance sources are used together, they introduce a challenge for reasoning about the origins of information. One approach maintains the diversity in the metadata, deferring the reconciliation of semantics till the resolution of queries. An alternative strategy utilizes explicit *provenance integration* to merge the metadata into a unified form, as long as the sources have compatible semantics and the same level of abstraction. We report on the case where a single underlying system is monitored at multiple vantage points, giving rise to the need for provenance-*only* integration.

2. Basic Integration

The Open Provenance Model [11] handles the problem of representing records from diverse domains by using graphs with all the application-specific semantics retained only as annotations on vertices and edges. This allows us to reduce the provenance-only integration challenge to a graph optimization problem.

Without loss of generality, we consider the problem of integrating two graphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$, given an arbitrary predefined *threshold of matching*, τ.

The matching between a vertex $v_1 \in V_1$ and another vertex $v_2 \in V_2$ is defined as:

$$m(v_1, v_2) = \sum_{a_i \in (A(v_1) \cap A(v_2))} w(a_i)$$

where $A(v)$ is the set of annotations a_i on the vertex v, and $w(a_i)$ is the weight accorded to a matching of annotation a_i – that is, the occurrence of a_i in the set of annotations $A(v_1)$ on vertex v_1 as well as in the set of annotations $A(v_2)$ on vertex v_2.

During graph integration, the intermediate graph $G_I = (V_I, E_I) = G_1 \uplus G_2$ will contain a single vertex $v = v_1 \uplus v_2$ instead of $v_1 \in V_1$ and $v_2 \in V_2$ if $m(v_1, v_2) \geq \tau$, where $A(v) = A(v_1) \cup A(v_2)$

contains all the annotations from the pair of vertices being integrated. ($\uplus$ denotes integration.) Similarly, if $v' = v_3 \uplus v_4$, where $v \in G_I, v_3 \in G_1$, and $v_4 \in G_2$, then an edge $e = (v, v') \in G_I$ will be defined in the intermediate graph. The annotations on the edge e will contain all the annotations on the edges $e_1 \in G_1 = (v_1, v_3)$ and $e_2 \in G_2 = (v_2, v_4)$, if one or more of e_1 and e_2 exist – that is, $A(e) = A(e_1) \cup A(e_2)$.

The combined integrated graph G_C will contain the intermediate graph G_I and all remaining unintegrated vertices and edges in the source graphs G_1 and G_2 — that is, $G_C = (G_1 \uplus G_2) \cup G_r$, where $G_r = (V_r, E_r)$ is the residue graph with $V_r = (V_1 \cup V_2) - V_I$ and $E_r = (E_1 \cup E_2) - E_I$. It is worth noting that if $\tau = \infty$, then $G_C = G_1 \cup G_2$ — that is, if the threshold of matching is too high, the combined graph will just be the two original graphs without any integration. It is also worth noting that if $\tau = 0$, then G_C will contain just one vertex v with all the source annotations – that is:

$$A(v) = \bigcup_{v_i \in (V_1 \cup V_2)} A(v_i)$$

Provenance graphs differ from data flow graphs in an important aspect, which is that each process or data artifact vertex is under the influence (or owned) by an agent. This results in a set of *integration constraints*, that limit whether two vertices under different influences can be combined. These constraints are formulated as costs – that is, when matching two vertices $v_1 \in V_1$ and $v_2 \in V_2$, the associated cost $\zeta(v_1, v_2)$ is based on the relationship between their owners $\Omega(v_1)$ and $\Omega(v_2)$, respectively. By definition, $\zeta(v_1, v_2) = 0$ if $\Omega(v_1) = \Omega(v_2)$. If the owners are from the same group, a low cost is imposed for integrating the vertices. If the owners are completely unrelated, $\zeta(v_1, v_2) = \infty$, which will prevent the vertices from being integrated unless the matching threshold is 0.

Ideally the cost of integration would be 0 – that is, only subsets of each graph under the influence of the same agent are integrated. In practice, due to the heterogeneity in sources from which provenance records arrive, parts of graphs that come from related owners can be merged with an associated *integration cost*, where $v = v_1 \uplus v_2$:

$$\zeta(G_1, G_2) = \sum_{v \in G_I} \zeta(v_1, v_2)$$

Intuitively, the more we are willing to trust that different owners' provenance records can be merged, the higher we can set the *trust tolerance* Υ. Since provenance graphs may be used for precision-sensitive applications (such as system diagnostics or identifying the source of anomalous activity after an intrusion), we must ensure that integration only occurs as long as the cost remains below Υ. Provenance graph integration can therefore be formulated as the optimization problem of finding the minimum threshold τ where the integration cost $\zeta(G_1, G_2)$ remains below the trust tolerance Υ:

$$\min \tau, \zeta(G_1, G_2) \leq \Upsilon$$

Integration of Heterogeneous Provenance

In the formulation above, two vertices from source graphs could be integrated into a single vertex if their match exceeded a predefined threshold. This corresponds to consuming the matching "budget" uniformly (or homogeneously) across all candidates. In an alternative formulation, the matching "budget" can be consumed heterogeneously, with some vertex pairs being better matched while other pairs being less matched, as long as the aggregate matching benefit when summed over all pairs exceeds a predefined threshold of matching. This formulation is harder to optimize since more permutations must be analyzed but is more useful in practice since it corresponds to the situation where there is diversity in the semantics of the vertex pairs in the graphs being integrated.

Case Study: Speech Processing

We now describe the first of two motivating case studies for provenance-only integration that we have encountered in practice. This case occurs when recording the details of speech processing workflows. Global data provenance from one source is combined with temporal overhead measurements from another. Together they allow a distributed application to be profiled to identify bottlenecks in input and output operations.

The NIGHTINGALE project [12] aimed to let monolingual users query information from newscasts and documents in multiple languages. Input data is transformed multiple times for automatic speech recognition, machine translation between languages, and distillation to extract responses to a query. The project used Berkeley Customs [4] Grid middleware to manage distributed computations running on more than 1,000 processors, accessing thousands of files that ranged in size from a few kilobytes to gigabytes, with a total of one petabyte of data.

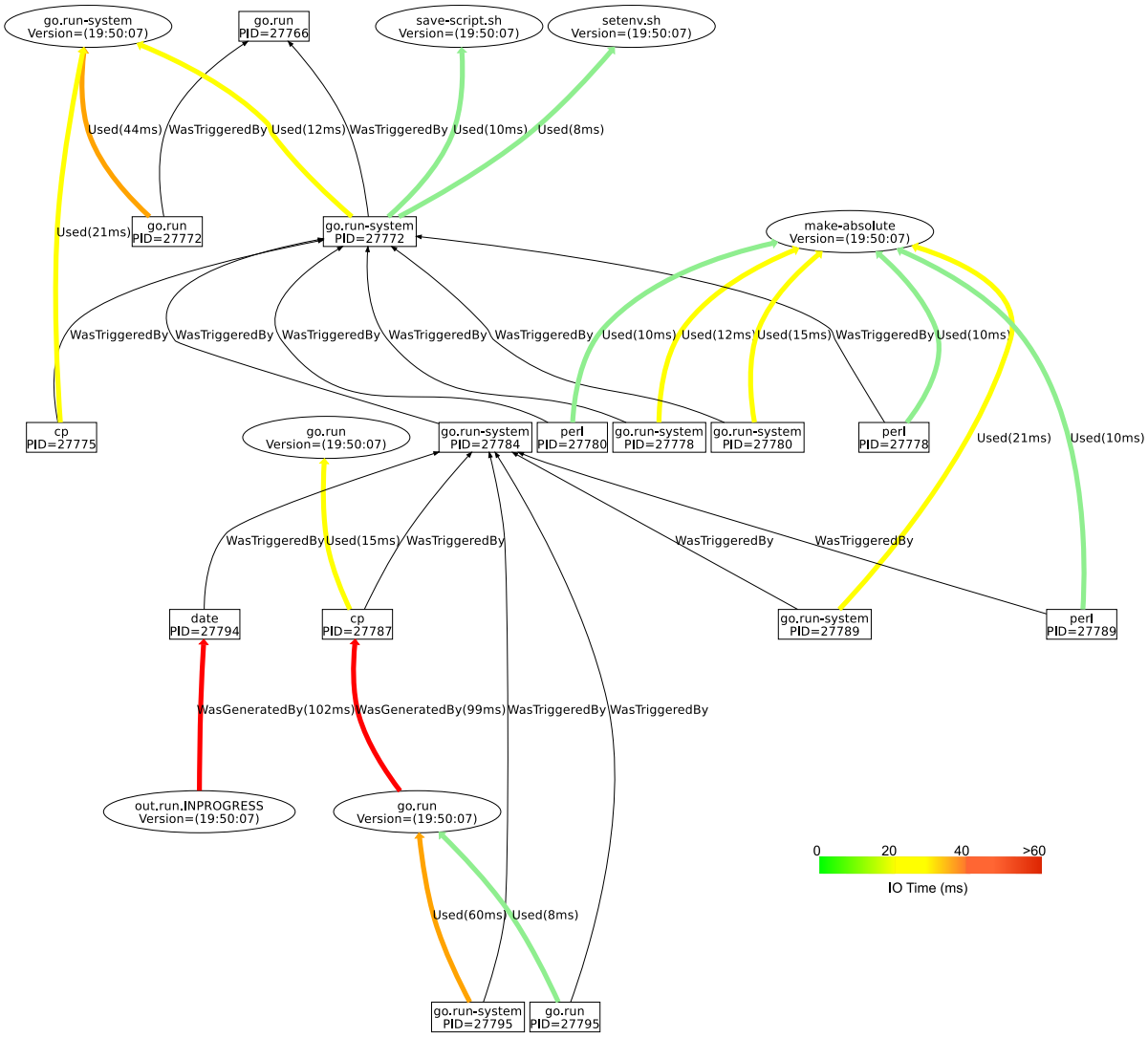

Figure 1. Application-specific temporally-enriched provenance records facilitate profiling. However, the high overhead of collecting these records prevents it from being done on a global basis. Instead these records are integrated with system-wide provenance from an alternative source. The combination allows resource-intensive parts of the computation to be identified, as visualized with this "heat map".

A typical computation run by a speech researcher executes hundreds of scripts and binaries on hundreds of processors that read and write thousands of files. Consequently, substantial amounts of time are spent on input and output (I/O). The scientists are therefore particularly interested in optimizing the workflow with respect to I/O.

One source of provenance is the operating system's audit trail, from which the file dependencies of all processes can be reconstructed. These records are collected across the entire system, for all processes executing and every file accessed. This results in a graph $G_1 = (V_1, E_1)$. Consequently, the level of detail that can be recorded is limited to manage the runtime overhead introduced.

Since the temporal overhead of input and output is of particular interest, a second source of provenance is used to obtain this information. This is possible in practice by limiting the collection of these records to a subset of system activity. Information gathered with system call interposition allows high fidelity monitoring. Its overhead is minimized by activation on just the part of the filesystem where applications of interest store their data. This results in a graph $G_2 = (V_2, E_2)$.

In previous work [8], we used an *integration policy* to specify how provenance elements from disparate sources relate. The current model allows integration to be defined by specifying only the trust tolerance Υ.

In this case study, annotations are unweighted. This is captured by using $w(a_i) = 1$ for all annotations $a_i \in A(v_1) \cup A(v_2)$, where $v_1 \in V_1$ and $v_2 \in V_2$. Therefore the match $m(v_1, v_2)$ is the number of annotations shared by v_1 and v_2.

We use $\Upsilon = 0$ to limit integration to pairs of vertices v_1 and v_2 with the same owner – that is, $\Omega(v_1) = \Omega(v_2)$. The integration process starts with a threshold of matching $\tau = 0$. This allows arbitrary pairs of vertices to be matched, but results in an integration cost $\zeta(G_1, G_2) > 0$ (since some pairs of vertices have different owners). τ is increased repeatedly until the integration cost $\zeta(G_1, G_2)$ drops to $\Upsilon = 0$.

The integration completes when $\tau = 2$ since this is the lowest value for which $\zeta(G_1, G_2) \leq 0$. This is because corresponding process vertices from G_1 and G_2 have matching process and group identifiers, and corresponding artifact vertices have matching file paths and modification times. No other annotations are shared.

The integration results in a provenance graph with the global coverage of the first source and the temporal fidelity of the second. Together they allow the identification of bottleneck "hot" areas in a distributed computation, as illustrated in Figure 1.

3. Fast Integration

Section 2 described a basic framework for integrating data provenance. In practice, the characteristics of the provenance may allow further simplification of the graph. In the basic framework, once a vertex $v_1 \in G_1$ is matched to another vertex $v_2 \in G_2$, the combined vertex $v = v_1 \uplus v_2$ is eliminated from consideration for further integration. However, other vertices in G_2 may have also been matches. Indeed, other vertices in G_1 may be similar enough that they can be represented by v. Consequently, the provenance-only integration can use an alternative algorithm that combines all vertices that are similar enough in a single fell swoop.

Combining the above simplification with basic integration has an important consequence. It eliminates the implicit step of identifying the specific pairs of vertices $v_1 \in G_1$ and $v_2 \in G_2$ that should be considered as candidates for matching. Optimally identifying such pairs is equivalent to solving the subgraph isomorphism problem, which is NP-hard [3]. Only in the rare case that the provenance graphs are planar can this problem be solved efficiently [7].

Algorithm 1

Input: $G_1 = (V_1, E_1), G_2 = (V_2, E_2), (\tau_v, \tau_e), \Upsilon$

```
1:  V ← V1 ∪ V2
2:  E ← E1 ∪ E2
3:  M ← ∅
4:  for v_i ∈ V do
5:      V_i ← {v_i}
6:      for v_j ∈ V do
7:          if match(v_i, v_j, τ_v, Υ) then
8:              V_i ← V_i ∪ {v_j}
9:      c_i ← integrate(V_i)
10:     M ← M ∪ {(c_i, V_i)}
11:     V ← (V − V_i) ∪ {c_i}
12: for e ∈ E do
13:     v_src ← source(e)
14:     v_dst ← destination(e)
15:     for (c_i, V_i) ∈ M do
16:         if v_src ∈ V_i then
17:             source(e) ← c_i
18:         if v_dst ∈ V_i then
19:             destination(e) ← c_i
20: for e_i ∈ E do
21:     E_i ← e_i
22:     for e_j ∈ E do
23:         if match(e_i, e_j, τ_e, Υ) then
24:             E_i ← E_i ∪ {e_j}
25:     f_i ← integrate(E_i)
26:     E ← (E − E_i) ∪ {f_i}
27: return G = (V, E)
```

In addition to the pair of provenance graphs that are to be integrated, the algorithm takes as input the threshold of matching τ and trust tolerance Υ that were described in Section 2. Empirical analysis motivated a refined definition for $\tau = (\tau_v, \tau_e)$ to allow different thresholds τ_v for vertex matching and τ_e for edge matching. Algorithm 1 describes how to perform optimized provenance integration.

The algorithm consists of three phases. In the first (on lines 4 to 11), all vertices that have at least τ_v identical annotations and no more than Υ dissimilar annotations if the owners differ, are combined into a single vertex. In the second phase (on lines 12 to 19),

any edge with a source or destination vertex that was combined in the previous step is updated. The original vertex is replaced by the combined vertex. In the last phase (on lines 20 to 26), any pair of edges with τ_e identical annotations and no more than Υ dissimilar ones when the owners differ is combined into a single edge. Note that *integrate()* takes a set of vertices or edges as input and emits a single combined vertex c_i or fused edge f_i, as described in Section 2.

Case Study: Intrusion Detection

Our second case study is that of intrusion detection, where sensors may be deployed at multiple locations on a host that is being monitored. Each sensor will collect different types of information. For example, on Android devices one source describes intra-application activity while another source tracks inter-application communication.

As the details in the logs increase and the granularity of monitoring becomes finer, the likelihood of detecting an intrusion increases. However, the storage overhead for retaining all the logs, the processing overhead for analyzing them, and the network overhead for distributed correlation all increase as well. To retain many of the benefits without the accompanying overhead, we can filter the logs as they are generated to extract and retain data provenance semantics. The provenance is represented as typed graphs. Sufficient information is present to identify users, processes, files, and network connections in the system.

When intrusion detection systems use automatically generated provenance records, they are faced with the challenge of integrating the information from multiple sources first. This is because different sensors contain provenance information at distinct levels of abstraction, have different levels of completeness, and use separate sets of identifiers to refer to the same concepts.

In the case of the Android mobile device platform, the audit subsystem in the kernel can be configured to emit a stream of events that allow the details of the agents, processes, and artifacts to be reconstructed, resulting in graph $G_1 = (V_1, E_1)$. These details are collected when system calls occur from applications to the operating system kernel. In contrast, the Android Binder framework monitors communication between applications and operates at a higher level of abstraction, resulting in graph $G_2 = (V_2, E_2)$.

To understand the need for integrating these two sources of provenance, we can consider a small example. When an application sends a text message, the audit log will only record the *ioctl()* system call. The fact that a text message was sent is lost since this information is only present in data structures that are passed by reference. The values of such arguments are not recorded since this would introduce significant overhead for system call auditing. Binder operates at the abstraction level of Android applications. It can therefore observe not only that a text message was sent but what the message was.

By combining one source's global view of the kernel's audit trail with another source's detailed visibility into inter-application communication, it is possible to create a more comprehensible reconstruction of system activity from the integrated provenance. Such an integrated view can be created with a trust tolerance of $\Upsilon = 0$, as in the speech processing case. In practice, ownership changes may be benign – for example, a file may be modified by multiple users. Such differences can be eliminated during integration by using higher values of Υ. However, this can have unintended side-effects as well. We defer further analysis till Section 5.

4. Implementation

SPADE [13] is SRI's open source data provenance middleware. Its provenance kernel supports the simultaneous use of multiple *reporter* modules. Each such module can collect and report provenance about an independent activity domain. The kernel filters these provenance streams before committing them to persistent storage. Previously, we have implemented *online* provenance integration as *filters* in the kernel. Such online integration is useful for stream processing, such as aggregating provenance elements into abstracted versions [8].

Stream processing uses finite buffers that limit the history available to an integration algorithm. In contrast, *offline* provenance integration can operate on provenance graphs in their entirety. To augment SPADE's online filtering capability, we developed an offline provenance integration utility that implements Algorithm 1. SPADE supports representing data provenance graphs in a fragment of the Graphviz project's DOT language [6]. SPADE's *Graphviz Reporter* takes as input a provenance graph in this language, and the kernel can be configured to emit output in this language with

```
    graph : digraph [ ID ] '{' stmt_list '}'
stmt_list : [ stmt [ ';' ] [ stmt_list ] ]
     stmt : node_stmt
          | edge_stmt
          | attr_stmt
attr_stmt : (graph | node | edge) attr_list
attr_list : '[' [ a_list ] ']' [ attr_list ]
   a_list : ID '=' ID [ (';' | ',') ] [ a_list ]
edge_stmt : node_id '->' node_id [ attr_list ]
node_stmt : node_id [ attr_list ]
  node_id : ID
```

Figure 2. Graphviz [9], created by AT&T Research for graph visualization, defines the DOT language [6]. SPADE supports the above fragment of the grammar for input and output. "Terminals are shown in bold font and nonterminals in italics. Literal characters are given in single quotes. Parentheses (and) indicate grouping. Square brackets [and] enclose optional items. Vertical bars | separate alternatives." [6] An *ID* can be an alphanumeric string or a numeral.

the *Graphviz Storage*. To integrate a pair of provenance graphs with the utility, they must be represented in the fragment of DOT shown in Figure 2.

We have used SPADE on Android for a number of purposes, including system diagnosis [10] and analyzing malware. Activity at the interface between applications and the operating system is recorded by the Android kernel's audit subsystem. SPADE uses this to collect system-wide provenance. However, interactions between applications occur through Binder, a user space message passing subsystem on Android. SPADE can treat this as a second source of provenance. Previously, whole system analysis required elements from the second source to be custom grafted into the provenance generated by the first. The provenance integration utility now allows this to be automated. We used a test workload generated by sending an SMS message on Android 4.2.1.

5. Evaluation

The need for provenance-only integration arises when a single underlying phenomenon is being reported on from multiple vantage points. Both of our case studies fall into this category. In this setting, the correctness of the integration can be judged by comparing the result to a reference model of the phenomenon. In our evaluation, we use knowledge of the semantics of the operating systems domain to validate the integration.

It is worth noting that provenance integration may also be used when independent data sets are combined. However, such integration is limited by the extent to which the semantics of the annotations are related. If the accompanying provenance records are judged to have compatible schema, the metric used to establish this can be applied to validate integration correctness.

Integration as Abstraction

We hypothesized that as graphs were increasingly integrated (using lower thresholds of matching), similar types of vertices would be collapsed together into unified vertices, effectively performing conceptual abstraction. We studied this by varying the threshold matching τ and looking for inflection points in the size of the integrated graph, shown in Figure 3, and the integration cost ζ, shown in Figure 4.

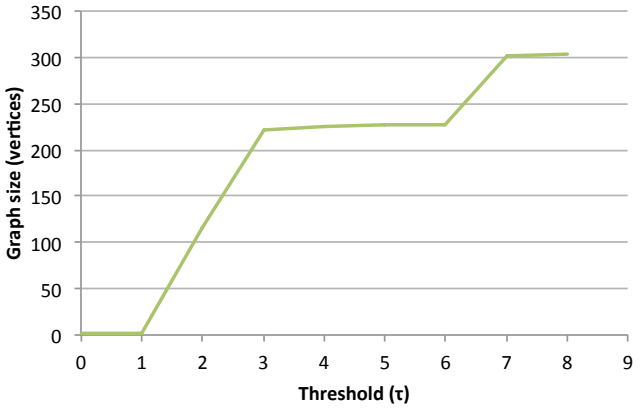

Figure 3. A lower *threshold of matching* τ allows vertices with fewer common annotations to be combined together. This results in a smaller provenance graph with fewer distinct vertices.

A high threshold of matching ensures that no abstraction occurs. Only vertices that are very similar can be combined into a single vertex in the integrated graph. When the threshold of matching τ is 8 (which is the maximum number of annotations), no integration is seen in Figure 3.

As τ drops to 6, vertices corresponding to different threads in a single process are combined together. Note that process vertices initially have eight annotations for the vertex type, program name, owner, owner group, process identifier, parent process, thread group, and the command line used to invoke the program.

When τ drops below 3, vertices that represent different versions of the same file are combined together.

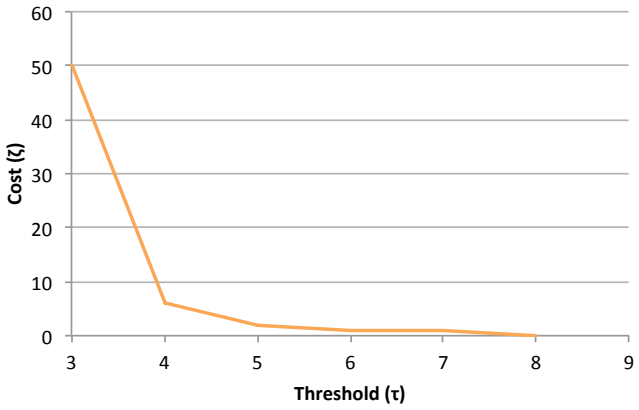

Figure 4. The *integration cost* ζ drops as fewer vertices with different owners are combined together (when the threshold of matching τ is increased).

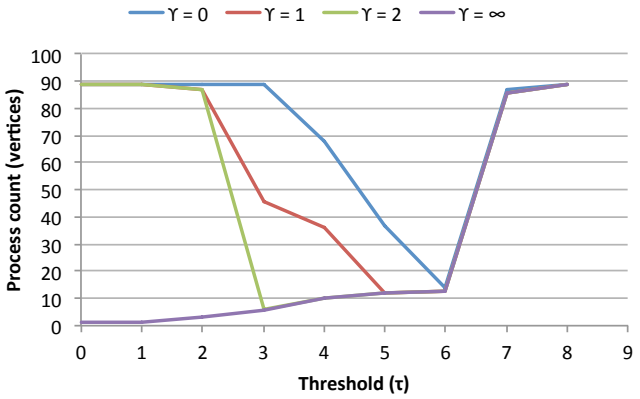

Figure 5. As the threshold of matching τ increases, more vertices can be integrated resulting in fewer distinct ones. When the tolerance of trust Υ is increased more vertices with different owners can be integrated, resulting in fewer distinct vertices after integration (for a given value of τ).

Note that artifact vertices initially have three annotations for the vertex type, filesystem path, and last modification time.

Recall that the cost of integrating provenance graphs arises from the conflation of agents that control processes. As more vertices with different owners are combined together, the integration cost grows.

If the graph size shrinks but the integration cost remains the same, vertices with the same owner are being integrated (as seen when τ drops from 7 to 6 in Figures 3 and 4). As the threshold of matching τ drops below 6 in Figure 4, the integration cost starts to grow. When τ drops below 4, the cost increases significantly because unrelated process vertices are being combined.

Fidelity of Attribution

A significant reason for capturing, storing, and querying provenance is to be able to correctly attribute ownership of data and its antecedents. When integration conflates ownership information, the utility of the provenance decreases. This motivated us to study the effect of varying the tolerance to ownership conflation, as specified by Υ, when integrating provenance records.

In this case study, the ownership of a process is defined by its user, group, and thread group. If all three are identical, the cost of integrating process vertices is 0. As more ownership elements differ, the cost of integration grows. For example, if a pair of process vertices have different thread groups but the same user and group, the cost of merging them is 1. Similarly, if they have the same group but different users and thread groups, the cost of integrating them is 2.

Unlike the basic integration algorithm, which tries to discover an optimal threshold of matching τ, the fast integration algorithm takes τ as a parameter. A pair of process vertices will only be integrated if they have at least τ annotations in common and the cost of integration is no greater than the trust tolerance Υ.

When an infinite tolerance is used, the integration cost is effectively ignored. As τ increases, fewer process vertices can be matched. This is seen in the plot for $\Upsilon = \infty$ in Figure 5, where the number of distinct process vertices after integration grows with τ.

Using $\Upsilon = 0$ specifies that no loss of fidelity of attribution information will be tolerated. So only vertices and edges with non-ownership information that is close enough (as specified by τ) can be integrated. As τ increases (for a given Υ), more vertices can be matched, resulting in fewer distinct vertices after integration. Note that the drop in the resulting number of vertices always has a floor below based on the count when $\Upsilon = \infty$ (since at this point the matching cost does not matter).

When we are willing to tolerate lower fidelity attribution, we can obtain more aggressive provenance-only integration. As Figure 5 illustrates, in practice it is best to select the lowest tolerance Υ that yields an acceptable level of integration. This will maintain the maximum fidelity of attribution.

6. Related Work

The need to integrate data arises in a wide variety of contexts. It has given rise to an extensive literature on techniques to address the problem in various settings, such as relational databases, the semantic web, and logic programming environments. Doan et al. [5] cover a broad set of approaches in their text on the topic, including some that apply to integrating data provenance.

Provenance metadata differs from generic data in multiple respects. It is structured, may be manually generated, often arrives in-order and is only appended to. These and other properties provide both challenges and opportunities for integrating provenance metadata. In earlier work, we have studied policy-based approaches for provenance integration [8]. Others have studied it in the context of the semantic web [14], Grid computing [15], interoperability of systems [1], and sharing provenance across organizations [2].

Our provenance-only work aims to be agnostic to the system domain, provides an optimization algorithm to effect the integration, and evaluates integration as a means of abstraction.

7. Conclusion

We described a framework for integrating provenance only. The need for this arises when multiple provenance traces (with different properties, such as temporal fidelity, monitored aspects, or abstraction levels) are collected for the same underlying phenomenon. We provided a provenance-only integration algorithm that merges provenance elements (vertices or edges) with sufficiently similar annotations (as defined by user-specified thresholds). It seeks to avoid integrating elements with different owners by imposing a cost for doing so. By finding the lowest thresholds that suffice for a given cost, provenance-only integration can be viewed as an optimization problem.

Acknowledgments

We thank the reviewers for their useful comments.

This material is based upon work supported by the National Science Foundation under Grant 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] Elaine Angelino, Uri Braun, David Holland, Peter Macko, Daniel Margo, and Margo Seltzer, Provenance integration requires reconciliation, *3rd USENIX Workshop on Theory and Practice of Provenance*, 2011.

[2] David Allen, Adriane Chapman, Barbara Blaustein, and Len Seligman, Getting it together: Enabling multi-organization provenance exchange, *3rd Workshop on Theory and Practice of Provenance*, 2011.

[3] Stephen Cook, The complexity of theorem-proving procedures, *3rd ACM Symposium on Theory of Computing*, 1971.

[4] Customs, `http://www.icsi.berkeley.edu/ftp/pub/ai/stolcke/software/`

[5] AnHai Doan, Alon Halevy, and Zachary Ives, Principles of data integration, Elsevier, 2012.

[6] Graphviz DOT, `http://www.graphviz.org/content/dot-language`

[7] David Eppstein, Subgraph isomorphism in planar graphs and related problems, *6th ACM-SIAM Symposium on Discrete Algorithms*, 1995.

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

[9] Graphviz, `http://www.graphviz.org/`

[10] Nathaniel Husted, Sharjeel Qureshi, Dawood Tariq, and Ashish Gehani, Android Provenance: Diagnosing Device Disorders, *5th USENIX Workshop on the Theory and Practice of Provenance (TaPP)*, 2013.

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

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

[13] Support for Provenance Auditing in Distributed Environments, `https://code.google.com/p/data-provenance/`

[14] Denise Umuhoza and Robin Braun, Trustworthiness assessment of knowledge on the semantic sensor web by provenance integration, *Trust Computing and Assurance Workshop*, 2012.

[15] Jing Zhao, Fan Sun, Carlo Torniai, Amol Bakshi, and Viktor Prasanna. A provenance-integration framework for distributed workflows in Grid environments, *Workshop on Grid and Utility Computing*, 2008.