Provenance in DISC Systems: Reducing Space Overhead at Runtime

Ralf Diestelkämper Melanie Herschel Priyanka Jadhav
IPVS - University of Stuttgart, Universitätsstr. 38, 70569 Stuttgart, Germany
{ralf.diestelkaemper|melanie.herschel}@ipvs.uni-stuttgart.de, st119882@stud.uni-stuttgart.de

Abstract

Data intensive scalable computing (DISC) systems, such as Apache Hadoop or Spark, allow to process large amounts of heterogenous data. For varying provenance applications, emerging provenance solutions for DISC systems track all source data items through each processing step, imposing a high space and time overhead during program execution.

We introduce a provenance collection approach that reduces the space overhead at runtime by sampling the input data based on the definition of equivalence classes. A preliminary empirical evaluation shows that this approach allows to satisfy many use cases of provenance applications in debugging and data exploration, indicating that provenance collection for a fraction of the input data items often suffices for selected provenance applications. When additional provenance is required, we further outline a method to collect provenance at query time, reusing, when possible, partial provenance already collected during program execution.

1. Motivation

Recently, data provenance based approaches have been proposed to analyze data and their processing in data intensive scalable computing (DISC) systems (Amsterdamer et al. 2011; Interlandi et al. 2015; Ikeda et al. 2011; Logothetis et al. 2013). Serving various applications such as data exploration, debugging, or monitoring, these approaches track each individual input data item during program execution, thus collecting fine-grained data provenance in an eager way. This introduces a significant space overhead, reported to typically be between 20% and 50% of the original input data size but that may also exceed 120% in some cases.

Clearly, methods to reduce the space overhead are necessary to make provenance collection and processing more attractive in real-world applications. First solutions include the use of approximated summarizations to reduce readily collected provenance data (Ainy et al. 2015) or provenance destillations (Alper et al. 2013) that aggregate raw workflow information for workflow publishing. While both solutions reduce the space needed to store provenance, they do so *after* the full-size provenance has been collected during program execution. Hence, they do not reduce provenance space and runtime overhead during program execution. The only system that we are aware of that aims at reducing provenance while it is processed in DISC systems is targeted towards streaming data (Glavic et al. 2013). The reduction techniques are tailored to series of recurring values.

This paper focuses on reducing the *space overhead* of provenance collection *at runtime* from the angle of the application that the collected provenance will eventually be used for. Tracking the full fine-grained provenance at runtime may be necessary in applications like auditing, security, or repeatability. However, our evaluation indicates that it is not required in a significant number of scenarios in other applications including data exploration, debugging, or monitoring. For these, collecting a small fraction of the provenance often suffices to obtain the same insights as with full provenance. The following example illustrates this rationale and will also serve as a running example.

Example 1.1 Given two datasets on radiation data and weather data (see Fig. 1(a) and (b), respectively), assume an analyst wants to determine if the weather condition influences the average radiation dose in Tokyo. The Spark program that aggregates the radiation and weather conditions and then joins them is shown in Fig. 1(c). To aggregate the weather conditions, the analyst makes use of a custom aggregator, which returns the predominant weather condition of the day. Based on the program result shown in Fig. 1(d), the analyst first notices that the average radiation on rainy May 1st is significantly higher ($> 65\%$) than on the clear day of May 2nd. This can naturally be explained by nuclear fallout. Maybe more interestingly though, he observes a suspiciously high radiation dose on May 3rd. By employing backward tracing, the analyst finds out that t_8, t_9, and t_{10} cause the high value, due for instance to a new sensor

TaPP 2017, June 22-23, 2017, Seattle, Washington.

	Date	City	Radiation	Unit	DevID	Type
t_1	11-05-01	Tokyo	0.052	μGy	1	m1
t_2	11-05-01	Tokyo	0.052	μGy	2	m1
t_3	11-05-01	Tokyo	0.051	μGy	3	m1
t_4	11-05-01	Kyoto	0.047	μGy	11	m1
t_5	11-05-01	Kyoto	0.047	μGy	12	m1
t_6	11-05-02	Tokyo	0.031	μGy	1	m1
t_7	11-05-03	Tokyo	0.041	μGy	1	m1
t_8	11-05-03	Tokyo	372	CPM	101	m2
t_9	11-05-03	Tokyo	370	CPM	102	m2
t_{10}	11-05-03	Tokyo	372	CPM	103	m2
t_{11}	11-05-03	Tokyo	0.041	μGy	2	m1

(a) Input table $Radiation$

	Date	Time	City	Temp	Condition	Wind
t_{12}	11-05-01	00:00	Tokyo	18	rainy	2
t_{13}	11-05-01	06:00	Tokyo	17	rainy	2
t_{14}	11-05-01	12:00	Tokyo	19	rainy	3
t_{15}	11-05-01	18:00	Tokyo	19	rainy	4
t_{16}	11-05-02	00:00	Tokyo	18	cloudy	4
t_{17}	11-05-02	06:00	Tokyo	15	clear	3
t_{18}	11-05-02	12:00	Tokyo	19	clear	3
t_{19}	11-05-02	18:00	Tokyo	19	clear	3
t_{20}	11-05-03	00:00	Tokyo	16	clear	4
t_{21}	11-05-03	06:00	Tokyo	16	clear	3
t_{22}	11-05-03	12:00	Tokyo	18	clear	2

(b) Input table $Weather$

```
1 radiation = radiation.filter($"City" === "Tokyo")
2 val dailyRad = radiation.groupBy("Date", "City")
3                 .mean("Radiation")
4 val dailyCond = radiation.groupBy("Date", "City")
5                 .agg(dailyWeather("Condition"))
6 val result = dailyRad.join(dailyCond, Seq("Date", "City"))
7 result.collect().foreach(println)
```

(c) Program code

	Date	City	Radiation	Condition
t_{23}	11-05-01	Tokyo	0.052	rainy
t_{24}	11-05-02	Tokyo	0.031	clear
t_{25}	11-05-03	Tokyo	222	clear

(d) Result data

Figure 1: Running Example: the input data residing in two tables $Radiation$ (a) and $Weather$ (b) is processed using the program in (c) to compute the average radiation dose in Tokyo depending on the weather conditions. The result is shown in (d)

that provides radiation values in a different measurement unit. Given this full provenance for tuple t_{25}, the analyst had three examples all pinpointing to the same cause of the problem. Most likely, one example would have been enough for the analyst to explain the high value in the result. ■

Based on the above rationale, this paper proposes an approach that reduces the space overhead of provenance collection at runtime by avoiding, from the start, the collection of what can be expected to be irrelevant or redundant provenance from the perspective of the application the provenance is used for. As there is no guarantee that users will never ask questions requiring provenance that has not been collected, we further outline a provenance recovery method that, when possible, reuses readily collected provenance to determine the queried provenance. In summary, our contributions are:

- A sampling based technique to reduce provenance collection during DISC program execution (Sec. 2).
- A preliminary comparative evaluation of the proposed provenance reduction technique, three further baseline sampling algorithms, and the state of the art full provenance collection (Sec. 3).
- A sketch of a provenance recovery algorithm that, when possible, leverages the provenance collected at program runtime to possibly accelerate query-time provenance computation (Sec. 4).

2. Provenance space overhead reduction

To reduce the space overhead incurred by provenance collection during DISC program execution, we pursue a pragmatic approach that, instead of tracking provenance of all input data items of a given dataset D, only tracks provenance of a representative sample of data items in D.

Definition 2.1 (Sample-based reduction) *Given a program P, a dataset $D = \{t_1, \ldots, t_n\}$ of data items t_i with schema $schema(t_i)$, and application constraints C, determine a set of data items $T \subseteq D$ for which to collect fine-grained provenance for the satisfaction of all constraints in C.*

In the above definition, D is not limited to a single relation, e.g., in our example, D contains all tuples from both the $Radiation$ and the $Weather$ relations. Furthermore, the definition of application constraints is deliberately left open, as it may in general be a constraint over any part of the provenance application, e.g., the maximum possible space overhead, properties of the algorithm such as provenance recoverability, or constraints that are based on provenance query expressiveness. In this paper, the first two examples apply, a more concise definition, specification, and analysis of constraints is part of future work.

The algorithm implementing a sample-based reduction presented in this section is based on the definition of equivalence classes, determined over a set of attributes.

Definition 2.2 (Equivalence classes) *Two data items $t_i, t_j \in D$ are equivalent wrt a set of attributes $A \subseteq \bigcup_{t \in D} schema(t)$ if the the projections of t_i and t_j over attributes A are equivalent. Equivalence classes are formed by applying the transitive closure over all pairs of equivalent data items $(t_i, t_j), i \neq j \in D \times D$.*

The notion of equivalent data items in the above definition is general enough to cover simple cases such as value equality to more complicated cases such as value similarity, synonyms, etc. In our current implementation, equivalent data items are data items that are equal in all attribute values of attributes in A. Selecting the best subset A of considered attributes is also left open for future research. Currently, we include all attributes referenced in the program P in A.

Algorithm 1: $equivalenceClassSampling(P, D)$

```
   Input: program P, dataset D
   Result: set T of data items to track
1  TMap ← ∅
2  A ← ∅
3  executionPlan ← getExecutionPlan(P)
4  A ← extractAttributes(executionPlan)
5  for t_i ∈ D do
6  |  V ←values(t_i, A)
7  |  if V ∉ TMap then
8  |  |  TMap ← TMap.put(V, t_i)
9  |  else
10 |  |  t_v ← TMap.put(V)
11 |  |  TMap ← TMap.put (V,choose (t_v, t_i))
12 return valueSet(TMap)
```

We now introduce equivalence class based sampling to reduce the space overhead of provenance at runtime, summarized in Alg. 1. Lines 1 and 2 initialize the structures used by our algorithm, i.e., the set of considered attributes A and a map, where keys identify equivalence classes and the corresponding values include the data item tracked for each equivalence class. Both are initially empty. In Line 3, the execution plan of P is fetched, based on which we can easily extract the attributes of source data items it refers to (Line 4). In the loop starting in Line 5, we iterate over all data items in D and extract the values in t_i for any attributes in $schema(t_i) \cap A$ (Line 6). Whenever these values are not present in $TMap$, it means a new equivalence class has been encountered, triggering the insertion of a new key-value pair in $TMap$. Otherwise, the data item complies with an existing equivalence class, for which we then decide which data item to keep as a representative. Once all data items of D have been processed, the algorithm returns all tuples in the value set of $TMap$ as the final set of tuples T, for which provenance will then be tracked.

Example 2.3 As set A, the algorithm extracts $Date$, $City$, $Radiation$, and $Condition$ from P. The equal values in these four attributes yield the following equivalence classes.

$$\{\{\underline{t_1}, t_2\}, \{\underline{t_3}\}, \{\underline{t_4}, t_5\}, \{\underline{t_6}\}, \{\underline{t_7}\}, \{\underline{t_8}, t_{10}\}, \{\underline{t_9}\}, \{\underline{t_{11}}\},$$
$$\{\underline{t_{12}}, t_{13}, t_{14}, t_{15}\}, \{\underline{t_{16}}\}, \{\underline{t_{17}}, t_{18}, t_{19}\}, \{\underline{t_{20}}, t_{21}, t_{22}\}\}$$

Tracked data items returned as T are underlined, showing that we track 12 instead of 22 data items.

■

3. Evaluation

Given our equivalence class based sampling approach, we study two aspects: (1) How effectively can the captured provenance be used for different provenance applications and (2) How significant is the reduction of the space overhead of provenance capture at runtime. We first present baseline algorithms and details of our evaluation methodology before we summarize our results. Our implementation in Spark extends Titian (Interlandi et al. 2015). We reuse its forward and backward provenance tracing and extend it by different reduction techniques for data item tracking.

Baselines. We use three baseline sampling techniques:

- *Random.* We collect provenance for a randomly sampled subset of input data items. A sampling ratio r determines which fraction of D is randomly sampled. When not mentioned otherwise, we set $r = \frac{1}{3}$.
- *Clustering.* We apply k-means clustering over the input data and track provenance for all items in the n largest clusters. By default, $k = 6$ and $n = 1$.
- *Histogram.* For each distinct data item schema in D, we create a histogram with b bins for the value distribution of the non-distinct attribute with the largest active domain. We then collect provenance for all data items in the f largest bins (default values: $b = 10, f = 1$).

The state of the art method to compare to is the collection of the data provenance of all data items, which is effective independently of the application. The focus of the present evaluation is to study how robust a sampling technique is with respect to fulfilling the provenance application need.

Provenance applications and evaluation scenarios. The different provenance applications that we consider in our study are data exploration, debugging, data quality analysis, and system monitoring. While this is not an exhaustive list of possible applications of provenance, the results for these provenance applications already indicate the practicality of sampling based space overhead reduction techniques for provenance runtime collection. For the four provenance applications, we have defined twelve different provenance scenarios on two different datasets, i.e., the Safecast radiation dataset[1] containing roughly 8 million data items and the US domestic flights dataset[2] with approximately 10 million data items. Each scenario covers either one or two of the aforementioned provenance applications, while each provenance application is covered by at least two scenarios.

A scenario (D, P, R, Q, QA) consists of an input dataset D, a Spark program P (typically contains analytical operators like filters, combinations of multiple items, and aggregations), a subset R of the result of P over D that raises a user question Q (e.g., high value is unexpected, raising the question why this value is so high), and an answer QA to question Q that we can obtain based on provenance (e.g., the high value is associated to a specific measurement unit).

Application-dependent effectiveness. Clearly, across all scenarios, tracking full provenance always allows users to get the answer QA to Q. The results obtained when applying sampling based approaches are summarized in Tab. 1. For each evaluated provenance application, it shows, by means of filled fractions of small pies, how frequently QA could be found for Q given the tracked provenance only.

[1] http://blog.safecast.org/data

[2] https://www.transtats.bts.gov/DL_SelectFields.asp?Table_ID=236

	Data Exploration	Debugging	Data Quality	Monitoring
Equivalence Reduction	●	◕	●	●
Random Sampling	◕	◕	◕	●
Clustering	◕	◔	◑	●
Histogram Reduction	◑	○	◑	●

Table 1: Use Cases

Scenario	Program output
1	The ten highest radiation values per year from 2010 to 2016
2	The average radiation level in the U.S. per month
3	A conversion table for the average radiation value for all units used in the dataset
4	The average arrival delay per year for each airline
5	The mean departure time based on the day of the week
6	The average arrival delay for each airport between the years 2002 and 2006

Table 2: Scenario overview

Overall, the equivalence based reduction approach satisfies almost all evaluation scenarios. When we apply random sampling, we are still able to answer a majority of evaluation scenarios. Only in scenarios in which the selectivity of the program P is very high ($< 1\%$), no answer is obtainable. When we employ the clustering based sampling technique, we recover provenance in our monitoring scenarios and in three out of four data exploration scenarios. For the one data exploration scenario that could not be answered, we observe that all tuples that could have provided the answer QA to the provenance question Q are located in few clusters. However, for none of those clusters provenance was tracked. The same issue appears in three out of four debugging scenarios and in half of the data quality scenarios. Finally, histogram based sampling yields the least promising results. In most scenarios the items necessary to answer the provenance question Q are not among those being tracked initially, as they do not occur frequently enough among the tracked items. The approach provides decent results only for monitoring, in which runtime outliers occur due to heavy skew in the input data.

Space reduction at runtime. We now evaluate the overall space overhead in MB at runtime incurred by different sampling approaches and the state-of-the-art approach without sampling on top of Titian. That is, the measured space overhead includes both an overhead caused internally by Spark when setting up intermediate data structures for provenance and the overhead caused by the actual provenance data.

Our evaluation considers six scenarios, their programs being briefly described in Tab. 2. The remaining six scenarios of the twelve mentioned earlier are not considered, since they only differ in the user question Q, which does not affect the selection of tracked source data items.

Fig. 2 shows the space overhead reduction at runtime on the six scenarios, in percentage of the size of the overhead when the full provenance is tracked. The results from Fig. 2 show that the equivalence class based approach yields especially good results in Scenarios 3, 4, and 5. In these scenarios, the overhead falls below 20% of the original overhead. The reason is that the columns over which we form the

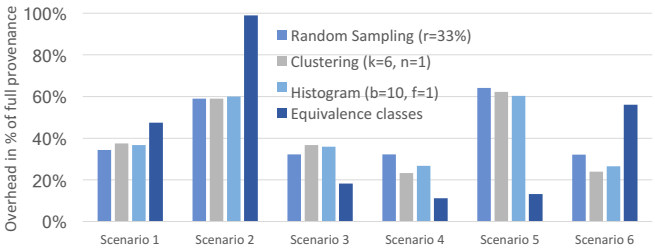

Figure 2: Space overhead evaluation

classes have comparatively few distinct values. For instance, in Scenario 5, we only consider the date column and the departure time, which have rather few distinct values. In contrast, Scenario 2 tracks almost all items, since the considered location and time attributes form an approximate unique column combination. Similarly, in Scenarios 1 and 6, we have to track around 50% of the input items, because the combination of columns yields many small equivalence classes.

We now consider the other three reduction techniques. In Scenarios 2 and 5, the total overhead is about 60% of the original one, while parameters were chosen such that roughly one third of the input data items is tracked. This high overhead can be explained by the fact that in these scenarios, the full provenance data itself is already negligible (only 9% and 5% of the size of D) compared to the space overhead produced internally by Titian.

4. Provenance recovery at query time

In this section, we outline an algorithm that leverages readily collected provenance when further provenance is necessary at query time. This case arises when a user question QA necessitates provenance that has not been captured during earlier processing. Note that the equivalence class based reduction technique is the only sampling techniques of those considered in this paper that allows such provenance recovery. This stems from the fact that its sampling is not solely based on characteristics of the data in D but also on information about P (i.e., the attributes A the program refers to).

Given a data item t_{out} that has not been (fully) tracked, the set of tracked result data items T_o, dataset D, and program P, the recovery algorithm proceeds as follows:

1. Find the set of output items $T_o^{t_{out}} \subseteq T_o$ that are causes for t_{out} not being tracked. An item $t_{o_i} \in T_o^{t_{out}}$ is a cause if the provenance of t_{out}, denoted $prov(t_{out})$, contains data items of D of the same equivalence class as tracked source data items in $prov(t_{o_i})$.
2. Trace back the provenance of data items in $T_o^{t_{out}}$ stepwise through P until either $traceBack(T_o^{t_{out}}) \subseteq prov(t_{out})$ or $traceBack(T_o^{t_{out}}) \supseteq prov(t_{out})$ can be verified. In the first case, the provenance to which the algorithm traced back can be added to $prov(t_{out})$ while in the latter case, we move to Step 3.

Date	Hospital	Patient
11-05-01	H1	P1
11-05-01	H1	P2
11-05-01	H2	P3
11-05-02	H1	P4

(a) $Hospitalizations$

	Date	Hospital	Patient	City	Radiation	Condition
t_{o_1}	11-05-01	H1	P1	Tokyo	0.052	rainy
t_{out}	11-05-01	H1	P2	Tokyo	0.052	rainy
	11-05-01	H2	P3	Tokyo	0.052	rainy
t_{o_2}	11-05-02	H1	P4	Tokyo	0.031	clear

(b) Result of joining $Hospitalizations$ with result in Fig. 1(d)

Figure 3: Extended example for provenance recovery

3. Trace forward the provenance of all data items to which back-tracing led and that are candidates for $prov(t_{out})$.
4. Among the provenance computed at the previous step, identify the subset to be included in $prov(t_{out})$.
5. Repeat Step 2 until no further back tracing is necessary to determine the complete $prov(t_{out})$.
6. Return $prov(t_{out})$ and store all provenance computed at Step 3.

Example 4.1 To illustrate the provenance recovery algorithm, we extend our running example. Assume that we have an additional table $Hospitalizations$, as depicted in Fig. 3(a). The result of joining this table by date to the result of Fig. 1(d) results in the table shown in Fig. 3(b). Using our equivalence class based sampling algorithm, only the first and last tuples, denoted t_{o_1} and t_{o_2} are tracked. For the second tuple, denoted t_{out}, the fact that it has not been tracked implies that source data it is based on did fall in the same equivalence class as other tracked data items. By analyzing P and the result data, we can identify that part of t_{out}'s provenance (the projection on $Date$, $City$, $Radiation$, and $Condition$) is the provenance of t_{23} in Fig. 1(d). Hence, this provenance can be added to $prov(t_{out})$ and there is no need to further trace back the program branch computing t_{23}. For the remaining attributes, program analysis reveals that we should trace back to tuples in $Hospitalizations$ where the date matches 11-05-01. Only one of the source items in $Hospitalizations$ having this value was initially chosen as representative of the equivalence class defined by this value. This is now compensated by tracking forward all tuples of this equivalence class. This will complete the provenance for the desired tuple t_{out} as well as the provenance of the third tuple in Fig. 3(b). ■

In the future, we plan to investigate the generalization of Step 1 of the above algorithm, i.e., under which circumstances can we identify the relevant result data items that originate from the same equivalence classes as t_{out}. A next research question is how to efficiently trace back through the collected provenance and to identify branches where further back tracing can be avoided (Step 2). For Step 3, methods to efficiently and effectively identify candidates for the provenance of t_{out} need to be developed. As a final note, our example shows that by combining the sampling based reduction technique with the recovery algorithm, we essentially may end up computing the complete provenance incrementally. Besides the question of a good tradeoff between overhead reduction at runtime versus runtime overhead at query time, this raises the question of how to combine our approach with reduction techniques that apply on the computed provenance (like (Ainy et al. 2015; Alper et al. 2013)).

5. Conclusion

We studied the problem of reducing the number of source data items for which provenance is eagerly collected while processing programs on DISC systems. We introduced a sampling method that selects data items to track based on equivalence classes. Our experimental evaluation indicates that, depending on the application for which provenance is collected, it is feasible to significantly reduce the provenance space overhead at runtime while still being effective with respect to the initial goal of the provenance application. We further sketched a method that reuses captured provenance to possibly speed-up provenance computation at query time, should an inspected result not have been tracked. As mentioned throughout the paper, the basic solutions presented in this paper open the path to a variety of research questions, which we intend to tackle in the future.

Acknowledgements. The authors thank the German Research Foundation (DFG) for supporting the project D03 of SFB/Transregio 161. We also thank Matteo Interlandi for sharing the Titian sources.

References

E. Ainy, P. Bourhis, S. B. Davidson, D. Deutch, and T. Milo. Approximated Summarization of Data Provenance. In *Conference on Information and Knowledge Management (CIKM)*, 2015.

P. Alper, K. Belhajjame, C. A. Goble, and P. Karagoz. Enhancing and abstracting scientific workflow provenance for data publishing. In *Extending Database Technology Workshop (EDBT)*, 2013.

Y. Amsterdamer, S. B. Davidson, D. Deutch, T. Milo, J. Stoyanovich, and V. Tannen. Putting Lipstick on Pig: Enabling Database-style Workflow Provenance. *Proceedings of the VLDB Endowment (PVLDB)*, 5(4), 2011.

B. Glavic, K. Sheykh Esmaili, P. M. Fischer, and N. Tatbul. Ariadne. In *Distributed Event-Based Systems (DEBS)*, 2013.

R. Ikeda, H. Park, and J. Widom. Provenance for Generalized Map and Reduce Workflows. In *Conference on Innovative Data Systems Research (CIDR)*, 2011.

M. Interlandi, K. Shah, S. D. Tetali, M. A. Gulzar, S. Yoo, M. Kim, T. Millstein, and T. Condie. Titian: data provenance support in Spark. *Proceedings of the VLDB Endowment (PVLDB)*, 9, 2015.

D. Logothetis, S. De, and K. Yocum. Scalable lineage capture for debugging DISC analytics. In *Symposium on Cloud Computing (SoCC)*, 2013.