Institute of Parallel and Distributed Systems

University of Stuttgart
Universitätsstraße 38
D–70569 Stuttgart

Master's Thesis Nr. MCS-0014

Workload based Provenance Capture Reduction

Priyanka Jadhav

Course of Study:	Computer Science
Examiner:	Prof. Dr. rer. nat. Melanie Herschel
Supervisor:	Ralf Diestelkämper, M.Sc.
Commenced:	August 1, 2016
Completed:	January 31, 2017
CR-Classification:	H.4

Abstract

Multiple solutions have been developed that collect provenance in Data-Intensive Scalable Computing (DISC) systems like Apache Spark and Apache Hadoop. Existing solutions include RAMP, Newt, Lipstick and Titian. Though these solutions support debugging within the dataflow programs, they introduce a space overhead of 30-50% of the size of the input data during provenance collection. In a productive environment, this overhead is too high to permanently track provenance and to store all the provenance information. That is why solutions exist that reduce the amount of provenance data after their collection. Among those are Prox, Propolis and distillations. However, they do not address the problem of incurring space overhead during the execution of a dataflow program. The existing provenance reduction techniques do not consider optimizing the provenance reduction based on particular use cases or applications of provenance.

The goal of this thesis is to find and evaluate application-dependent provenance data reduction techniques that are applicable during execution of dataflow programs. To this end, we survey multiple applications and use cases of provenance like data exploration, monitoring, data quality etc. In addition, we analyze how provenance is being used in them. Furthermore, we introduce nine data reduction techniques that can be applied to provenance in the context of different use cases.

We formally describe and evaluate four out of the nine techniques - sampling, histogram, clustering and equivalence classes on top of Apache Spark. There is no benchmark available to test different provenance solutions. Hence, we define six scenarios on two different datasets to evaluate them. We also consider the application of provenance in each scenario. We use these techniques to obtain reduced provenance data then, we introduce three metrics to compare the reduced provenance data to full provenance. We perform a quantitative analysis to compare different techniques based on these metrics. Afterwards, we perform a qualitative analysis to examine the effectiveness of different reduction techniques in the context of a particular use case.

Contents

List of Figures

List of Tables

1 Introduction

Modern data intensive applications collect information from multiple sources and aggregate as well as modify it in complex ways. Hence, it is almost impossible to understand the process semantics and the derivation of results. Provenance is helpful in this context. Provenance refers to the origin and evolution of a piece of data. It can also be called lineage, pedigree, parentage, genealogy, and filiation [SPG05]. Typical provenance information includes details on sources of data and computations performed to derive the data. Provenance can be associated not just with the data produced, but also with the process(es) that enabled the creation of the data itself.

In the age of the internet, data is made available online and is created, modified and copied by multiple entities of varying data quality and trustworthiness. Hence collecting provenance is important for many applications to assess the quality of data based on its sources and derivation process. Provenance can also be helpful in monitoring, debugging and troubleshooting as it includes information on the computations that were performed on data.

Recent Data-Intensive Scalable Computing (DISC) systems have to deal with huge quantities of data and perform analysis on them. Errors can occur in any component in the vast infrastructure supported by these systems. These systems, like Apache Hadoop [Shv+10], Apache Pig [Ols+08] and Apache Spark [The16] have very little support for debugging the processing logic [Int+15]. Most programmers have to spend hours in either collecting logs to find the faulty component or data, or engage themselves in trial-and-error debugging by replaying the task(s) with certain subset of data to obtain erroneous objects. With large input data sets, such debugging techniques become very cumbersome.

These DISC systems usually employ workflows for processing large data sets and employ workflow provenance techniques to track provenance. However, workflow provenance assumes that every output datum is derived from or dependent on all of the input data. This assumption is not always valid. In many cases, an output datum may not depend on all of its input data, but only on a certain subset of it. Hence during provenance collection, we must consider only the subset of input data that a given output record is based on. This provenance which captures dependencies between every output datum and its corresponding subset of input data is called fine-grained provenance. RAMP [IPW11], Newt [LDY13], Lipstick [Ams+11] and Titian [Int+15] provide provenance tracking frameworks for addressing fine grained dependencies in the data within DISC systems and are discussed further in Chapter 2

These solutions collect provenance when workflow is being executed and track fine grained dependencies whilst dealing with "big" data in DISC systems. That is why they collect and store additional information to track provenance in comparison to a workflow that does not track provenance. While all of these solutions track fine grained provenance and support debugging of workflows, they either introduce significant space overhead or do not scale for terabytes of data. Experimental evaluation shows that the size of the lineage increases proportionally with the size of the input dataset. More specifically, the lineage size is between 30% and 50% of the size of the initial dataset [Int+15]. In case of datasets larger than 90GB, the amount of lineage stored when tracking provenance in Spark workflows is close to 45GB [Int+15]. As a consequence, they are not suitable for permanent use in production environments.

One way to deal with the problem of high space overhead incurred by tracking provenance in DISC systems is by summarizing or approximating the provenance data. Some applications or use cases of provenance do not require provenance tracking for every single piece of input data. For example, in the case of using provenance for data quality, not all the input records may be used in measuring a data quality metric (e.g., accuracy, correctness), but instead a data sample can be used to provide insight on overall data based on the sample. Many applications which deal with large scale data, like Crowd Sourcing applications (E.g. IMDB, TripAdvisor) do not need to preserve information on every single data item present. Instead, the real value these applications provide is in the aggregated high level view of the data which is required for analysis [Ain+14]. Minor details can be ignored or summarized. Also, less provenance data paves way for faster provenance querying and less data maintenance activities.

There has been a lot of research work dedicated to reducing the collected amount of provenance using approximation or summarization. Prox [Ain+14], Propolis [DMT14], Provenance distillations [Alp+13] and Ariadne [Gla+13] are a few examples which are described in Chapter 2. However, all these solutions do not approximate or compress the provenance data when it is being collected. They operate on the provenance data after its collection. Hence, the size occupied by provenance is still significantly large to be allowed to use in production systems. Also, these solutions have not been tailored to a particular use case of provenance. Provenance is employed in many use cases, like monitoring, debugging, data quality etc. which are discussed in detail in Chapter 3. If the use case in which provenance can be used is known in advance, provenance collection can be optimized in a better way. Provenance data reduction can then be performed based on a particular use case of provenance.

The goal of this thesis is to identify existing and create novel data value reduction techniques for provenance data based on its use case. These techniques are applied during collection of provenance when the workflow is being executed. In Chapter 4, we formally describe these data value reduction techniques like sampling, deduplication etc. and in Chapter 5, we relate each of these techniques with their corresponding use cases enlisted in Chapter 3. The following are the research contributions provided by this thesis:

1. The first contribution of this thesis is to identify multiple use cases for provenance collection. We identify these use cases so as to develop use case specific provenance reduction techniques.
2. The second contribution is to find provenance reduction methods applicable during provenance collection. These techniques reduce the overall amount of provenance collected during execution of the workflow.
3. The next contribution is to provide a formal description for the reduction methods identified. In addition, we present the use cases where we can apply each data reduction technique.
4. In order to show the importance of data value reduction techniques, we perform an evaluation on top of Apache Spark. The focus of this evaluation is on space overhead. In addition, a comparison is made between different data reduction techniques based on how well they address the use case of provenance.

Structure of the thesis

The structure of the remaining thesis is organized as follows: In Chapter 2, we present the background necessary to understand this thesis. We introduce the two important ways in which provenance is captured and briefly describe the types of provenance. We further discuss the granularities in which the provenance data can be represented. Also, as part of this chapter, we present research work performed in the area related to the topic of this thesis and describe the existing solutions that track provenance in DISC systems. We then focus on the necessity of having approximation or summarization in provenance data and discuss the relevant research.

In Chapter 3, we classify the use cases and application areas of provenance. We present the broad categories like monitoring, debugging etc. and then describe each of these categories in detail.

In Chapter 4, we introduce nine data value reduction techniques for reducing the provenance data and formally describe four of them - Random sampling, Histogram, Clustering and Equivalence classes. Furthermore, we present the use cases to which these techniques can be applied.

In Chapter 5, we describe implementation of data value reduction techniques and information on experimental setup used to evaluate them. We further present the results obtained after the evaluation.

Chapter 6 completes the thesis with a conclusion and potential for future work.

2 Related Work

This chapter discusses concepts that help to understand the content of this thesis. Provenance collection and representation differs depending on the domain it is used in. In the following sections, we describe briefly the different types of provenance and different granularities in which provenance data is represented. Finally, we provide an introduction to two of the approaches employed to capture provenance. This chapter also covers work on provenance in Data Intensive Scalable Computing (DISC) systems. These systems execute processes with data at scale of tera-bytes and peta-bytes and have complex iterative logic. Hence collecting provenance within such systems is a challenging task. In the following sections, we present solutions that track provenance in DISC systems. However, these solutions produce large amounts of provenance data that incur space overhead. That is why we need solutions to reduce the amount of provenance data. Provenance approximation or summarization solutions are helpful in this context. Few of such solutions are discussed towards the end of this chapter.

2.1 Types of provenance

[HH16] classifies the types of provenance into the following four types - data provenance, workflow provenance, information systems provenance and provenance metadata. These four types form a hierarchy as shown in Figure 2.1. As we move up from Provenance metadata to data provenance, the more specific and specialized the provenance type becomes. That is, the set of possible processes the type of provenance can be applied to reduces. Provenance metadata deals with any kind of metadata which describes the origin and derivation of the data item under consideration and the transformations applied to it. This is the most generalized definition for provenance information. Information systems provenance collects provenance information adhering to some standard representation. This information contains metadata on the transformations which act on the data object we are interested in.

Workflows pose requirements like repeated running with different parameters, or with minor changes, which requires to store provenance information on the tasks within the workflow across its multiple runs. Workflow provenance is similar to information systems' provenance, that is workflow provenance just deals with one specific type of information system, the workflows. Data provenance includes details regarding individual data items. It deals with what operations every individual data item undergoes [HH16]. Data provenance is applied to structured data models and dataflow languages whose transformations have clearly defined semantics, like Spark [The16] and SQL [Her15]. This enables the re-engineering of each of the transformations where annotations are propagated across the process execution. The resulting data model of provenance

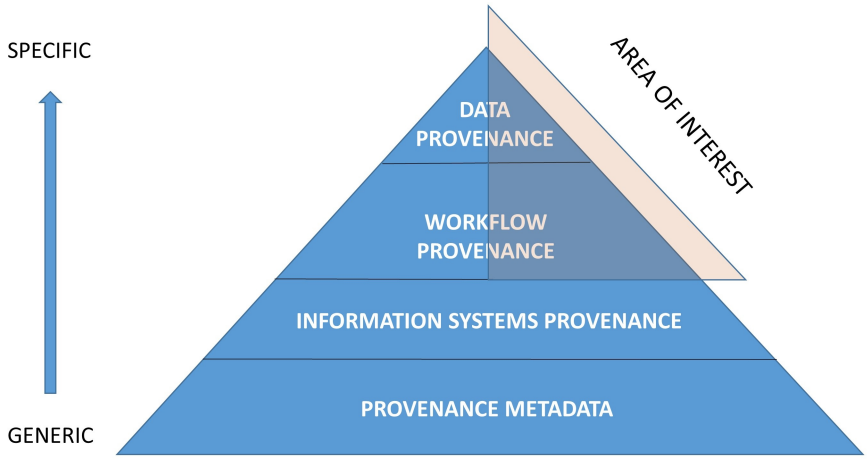


Figure 2.1: Types of Provenance [HH16]

data is closely related to the data model of the process and semantics of transformations of the dataflow language.

In Section 2.4, we describe data provenance capture solutions that are based on dataflow languages with clearly defined semantics like RAMP [IPW11], Lipstick [Ams+11] and Titian [Int+15]. These solutions extend transformations within the data manipulation language to pass on their provenance annotations as the process is being executed. This thesis focuses on data provenance and introduces provenance data reduction techniques for data provenance, however they can be extended to workflow provenance, too. In Figure 2.1, we highlight the area where the interest of this thesis lies - data provenance and workflow provenance.

2.2 Granularities of provenance

There are two granularities or the level of detail in which provenance data is categorized. Coarse-grained provenance does not track dependencies between output data and its corresponding input data, whereas fine-grained provenance does so for every individual data tuple in the output. Coarse-grained provenance employed mostly in workflow provenance systems, includes the complete history of the derivation of some dataset and contains a sequence of transformations executed to produce the same. In addition, it may also contain the interaction of data with external devices and people. Coarse-grained provenance collects provenance for a system where

each of its modules are considered as black boxes whose inner workings are not accessible or not relevant. Hence it is sometimes called “black box” provenance [CCBD06]. It contains sequence of module invocations, their logical flow and the input-output relations, hiding the intricate relations between individual data items. Coarse-grained provenance assumes that every output tuple of a module is dependent on every input tuple, which is not always the case.

Fine-grained provenance maintains the derivation process of every piece of data in a dataset. It does not assume that an output tuple is dependent on every input tuple, but instead on a subset of the input dataset. It gives a more detailed view on the relationship between output and input tuples of a module when compared to coarse-grained provenance. Fine-grained provenance can reproduce results which is hard in case of coarse-grained provenance. The former documents the source data as well as the process used to create certain output data. [BKWC01] provides an important distinction within fine-grained provenance. There is why- and where- provenance. Why provenance documents the source data items from where the result data is copied from and the reason on why the result data was produced. Where provenance only identifies the source data items that produced the result. Another important notion of provenance worth mentioning is how- provenance. How- provenance describes how a result data item is derived from its source data items.

In this thesis, we focus on fine-grained provenance and discuss multiple solutions developed to track fine-grained provenance in dataflow systems like Apache Spark and Apache Hadoop in Section 2.4.

2.3 Eager and lazy approaches in capturing provenance

Provenance is captured in different ways based on the application requirements. There are two approaches of capturing provenance - the eager approach and lazy approach. The former focuses on computing provenance information when data is created, while the latter computes provenance data when it is requested. In the eager approach, also known as annotation approach or book-keeping approach [SPG05], the transformations within the process are modified such that extra information is transferred from the source until the completion of the process. This extra information, or annotations can be represented in multiple ways, from unique identifiers for every data tuple to labels describing what transformation every data tuple undergoes. The transformations within the data process have a task of capturing, processing and carrying these annotations across the execution, in addition to the primary task they perform. Hence, the provenance of an output data tuple can usually be derived by inspecting the output data and the annotations captured.

In eager approach, provenance is pre-computed and readily available for use after the process execution. In contrast to this, provenance must be captured on-demand in the case of lazy approach. That is, provenance is computed as and when necessary by inspecting the input data, the output data, and the transformations present in the data processing flow. It does

not require modification of transformations within the process to carry additional information (annotations). Hence this approach is used when modification to data flow process is not possible. Lazy approaches use the concept of inversion to derive input data corresponding to certain output data. Examples of inversion operations can be queries and transformations which can be inverted to derive the input data based on the output data and intermediate data between the transformations of a process [ACT06; CWW00].

At runtime, the eager approach has higher overhead than lazy approach in terms of time and space, due to the processing of additional data (annotations) and storing the same. However, it has the advantage that if sufficiently detailed annotations are recorded, provenance may be computed directly from the output dataset and the annotations. Hence it allows capturing provenance without inspecting the source data [SPG05]. Thus, when sources are not always available, eager approach can be used. This thesis discusses on and presents eager provenance collection systems like RAMP [IPW11], NEWT [LDY13], TITIAN [Int+15] etc. in the next sections. Also, in the rest of this thesis, all the solutions that capture provenance use eager approach unless stated otherwise.

2.4 Data provenance in DISC systems

Data Intensive Scalable Computing systems were introduced to process enormous amounts of data in an efficient way. These systems have the capability of processing multi-terabytes of data in minutes. Apache Hadoop [Shv+10], Apache Pig [Ols+08] and Apache Spark [The16] are few examples of DISC systems. To process such huge data, we need to divide it into subsets and place them on multiple nodes (machines) so that we access them in parallel and improve the efficiency. These nodes together form a cluster computing system. Tracking provenance for processes running on a cluster in DISC systems is important. For example, Provenance can be helpful in debugging processes running on these systems. Errors can occur anywhere in this cluster of hundreds of nodes. There may be bugs due to semantics or syntax, due to unclean or invalid data or due to incorrectly functioning hardware. Tracking the lifecycle of a dataset during its transformation would help in debugging of processing flow in DISC systems.

Workflow Provenance techniques are applied to track the data in these systems. Workflow provenance assumes that every output is dependent on every input, and not on a subset of input data. When we assume that output depends on all of input data, we are restricting ourselves to obtaining a very coarse approximation of the data dependencies that are present in the workflow execution and that is not suitable for debugging. We need a mechanism to capture metadata describing lifecycle of every data tuple during process execution. Fine-grained provenance provides this information and involves tracking provenance for every data tuple in input. RAMP [IPW11], Newt [LDY13], Lipstick [Ams+11], Titian [Int+15] and Explaining outputs [Cho+16] each provide a provenance tracking framework for addressing fine-grained dependencies in the data within various DISC systems.

2.4.1 RAMP - Reduce And Map Provenance

MapReduce[DG08] paradigm is used in distributed data processing. MapReduce jobs can be executed on top of Apache Hadoop. Sometimes, the data processing logic cannot be represented and confined to a single MapReduce job and hence, multiple jobs are combined to form a MapReduce Workflow. MapReduce has two main functions – map and reduce. For every input data tuple, map function produces zero or one output tuples. Reduce function accepts key value pairs as input and groups input data based on the key. It produces either zero or more tuples as its output.

RAMP stands for Reduce And Map Provenance [IPW11] and tracks provenance in MapReduce workflows. It is developed as an extension to Hadoop and captures fine-grained provenance by building wrappers around map and reduce functions in order to capture provenance dependencies between input and output records. The provenance of an output tuple of map function is the input tuple that produced it. The provenance of an output tuple of a reduce function is the set of tuples (based on key) that produces the output tuple. RAMP uses the wrapping approach to add provenance tracking functionality to Hadoop components – mapper, reducer, record-reader, record-writer and combiner. This wrapping is performed such that it does not disrupt Hadoop's fault tolerance and parallel processing.

RAMP assigns unique identifiers to each of the data tuples and stores the mapping between output and input data tuples. Provenance in the form of these identifiers is captured one transformation/function at a time. These mappings when extended to consider all transformations within the workflow act as an end-to-end trace between the output data tuples and their respective input tuples. The authors tested RAMP with wordcount and terasort programs and found that provenance tracking incurs time overhead of 20-76% [PIW11] and space overhead of 19-21% for terasort [IPW11].

2.4.2 Newt - Scalable lineage capture for debugging DISC analytics

Newt was introduced as a general scalable architecture to capture fine-grained provenance in DISC systems. It was designed and developed such that it would be flexible enough to be used with multiple DISC systems like Apache Hadoop, Hyracks and Apache Spark. Newt uses the concept of instrumentation, i.e., modifying the behavior of transformations, to generate and propagate fine-grained provenance along process execution. It provides an instrumentation API that collects lineage from common transformations across various DISC systems. Newt requires manual instrumentation of DISC system in order to capture provenance; a developer must add instrumentation code in form of APIs into each actor. Actor is any entity that transforms data. For example, a MapReduce job is considered as an actor. These APIs treat the transformations as black boxes and store the inputs and outputs for the same. These associations between input and output for every transformation are stored in an association table. The APIs capture provenance of a workflow one transformation at a time.

Newt stores the association tables as relational tables in SQL and creates indexes on them to optimize the access. Being flexible, it provides a separate MySQL cluster where users can query the collected lineage data with SQL, irrespective of the DISC system used, be it Hadoop, Hyracks or Spark. Evaluation and demonstration of Newt framework was performed on Apache Hadoop and Hyracks and overhead incurred was 10-51% in terms of time and 30-120% in space [De12].

Though RAMP and NEWT capture fine-grained provenance, they do not let the user access intermediate data between transformations. The lineage data is queried using a separate programming interface different from the DISC system's interface. Hence it does not support interactive debugging sessions. Also, they use external storage to store lineage which restricts scalability when workflows access enormous data sets.

2.4.3 Titian - Data provenance support in Spark

To address the limitations mentioned above, there was a need for a system which could provide interactive debugging, and also the ability to trace data across transformations at interactive speeds. Titian [Int+15] is useful in this context and extends Apache Spark with step-by-step provenance tracking and providing access to intermediate data between operators. Titian materializes the dependencies between individual records in a Spark job (including the intermediate ones), and offers an API for interactive forward and backward tracing of dependencies. "Apache Spark is a fast and general engine for big data processing, with built-in modules for streaming, SQL, machine learning and graph processing" [The16].

Spark is based on Resilient Distributed Dataset (RDD) abstraction which are data subsets partitioned across the nodes of the cluster so that they can be operated on in parallel [Che+16]. This RDD abstraction provides a set of transformations to be performed on datasets. Spark maintains its own data lineage of these datasets, hence providing fault tolerance in the event of crashes etc. Titian introduces LineageRDD that extends regular RDD with provenance-capture functionalities. The LineageRDD enables tracing within processes running on Spark. It returns a reference to a dataset from where it is called and starts the tracing; all the native RDD transformations can be executed on this reference. Also, it can be used in interactive Spark sessions as it is built on the Spark runtime and programming model.

Figure 2.2 shows how tracing is performed in Titian. Titian generates unique identifiers for each new record, and associates output records for a given operation with its relevant input records. Associations between input and output record identifiers for every transformation are stored in a table called Association table. This table containing two-columns maps the unique identifier of an output tuple to unique identifier of its input tuple(s). The association tables for all the transformations are saved on the local Spark storage Layer called Block Manager. Tracing can be performed by recursively joining association tables saved in Block Manager.

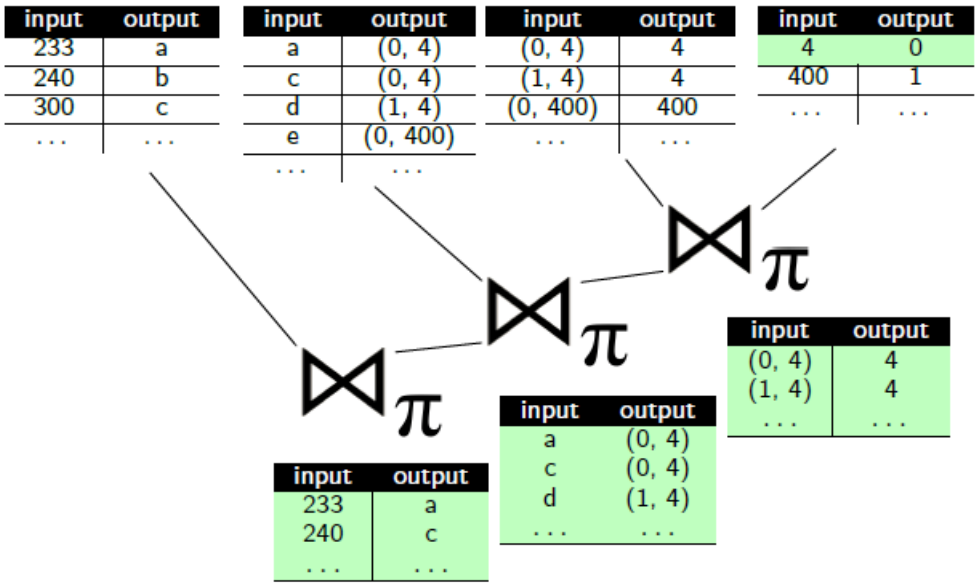


Figure 2.2: Tracing is performed by recursively joining the association tables containing unique record identifiers [Int+15]

Titian provides the following methods that help in interactive tracing – goBack(), goBackAll(), goNext() and goNextAll(). These can be used during execution to view dependencies between input and output records interactively. Given certain output record(s), goBackAll provides list of all record(s) which contributed to certain output record(s) across the execution of various transformations. goNextAll returns all the records from output dataset that depend on a set of certain input record(s). A single step in backward and forward tracing is provided by goBack or goNext methods respectively.

Evaluation performed on Titian concludes that lineage size is usually around 30% of the size of the input dataset. In some cases, it goes up to 50% and hence it is not always suitable for production environments. In terms of runtime overhead, Titian is never more than 1.3X Spark for datasets smaller than 100GB, and never more than 1.67X at larger dataset sizes [Int+15].

2.4.4 Lipstick - Enabling database style workflow provenance

As stated earlier, RAMP and Newt capture provenance which provides mapping between output data tuples with corresponding input data tuples. They do not provide information on intermediate data that flows between transformations within the process. Titian provides access to intermediate data that flows between stages(set of transformations), if not every transformation in a Spark process. Another work in the direction of providing access to intermediate data in DISC systems was performed by [Ams+11]. They "married" workflow provenance to data provenance in Pig Latin [Ols+08] modules on top of Apache Pig. Workflow provenance mainly considers coarse-grained dependencies, which assume that the output of a module is based on

all of its input. This assumption doesn't hold true for most of the applications, as the output may depend on a subset of inputs as well as the internal state of the modules. The internal state of the module is sometimes dependent on inputs from older executions, too. Hence, Amsterdamer et al. [Ams+11] proposed a framework which encapsulates these fine-grained dependencies as well as the internal state of the modules in tracking provenance.

Because we are dealing with fine-grained dependencies, the provenance data stored will be detailed and at the scale of input "big" data which can go upto tera-bytes and peta-bytes. Also the workflows contain large numbers of modules within them and will have multiple executions, adding to the already large provenance data. A novel form of representation called provenance graph was introduced to capture this detailed information. The workflow executions can be represented using these graphs at two visualization levels - coarse and fine. At coarse level, the invocations of the modules and their sequence are recorded. Also data flow at module level and logical flow between each of the modules are modelled as part of this graph. At the finer level, internal details such as the state of module database, operations performed in every module and computational dependencies between output and input data tuples are exposed. An important operation called deletion propagation is also supported by this framework. Deletion propagation helps to analyze how potential deletions propagate through the workflow execution, allowing users to assess the effect a certain tuple has on the generation of some other tuple(s).

A prototype called Lipstick with the following subsystems demonstrates the framework. **Provenance tracker** writes the output of the provenance annotated tuples to file system. The Pig Latin expressions within the modules are translated into bag semantics of Nested Relational Calculus [Bun+95] and annotated using the semiring [GKT07] annotations. **Query processor** reads this provenance annotated tuples from disk and creates the provenance graph. This provenance graph describes the How Provenance for the result data set. ZoomIn/ZoomOut and delete propagation queries are expressed as graph matching queries on this graph. The ZoomIn/ZoomOut queries allow users to query provenance information at both coarse and fine levels of granularities. Overhead is introduced when tracking provenance during the execution of the workflow. Evaluation performed showed that Lipstick introduces a time overhead of upto 35%. Lipstick makes use of parallelism provided by hadoop to provide scalability. The larger the size of the provenance graph, the larger is the time required to load it in memory.

2.4.5 Explaining outputs in modern data analytics

The work in [Cho+16] designs and implements a framework that explains (by collection of provenance) interactively, the outputs of modern data parallel computations and iterative data analytics. These explanations tells the analyst why and how a certain output record was produced from the input data. [IPW11], [LDY13], [Ams+11] and [Int+15] adopt naive backward tracing. According to [IPW11], A monotonic operator T satisfies the following condition: if for any input sets I_1, I_2 with $I_1 \subseteq I_2$, then $T(I_1) \subseteq T(I_2)$. Any map operator is monotonic, however not all reduce operators are monotonic. The naive backward tracing has the following issues:

1. They produce too much information that makes the explanation difficult to understand

2. There is not enough information that would allow for complete reproduction of the result when there are more than one non-monotonic operators in the dataflow.

These drawbacks do not allow the analyst or debugger to seamlessly debug the logical flow within the program. The authors provide a general framework that addresses these drawbacks. Their approach is based on Differential Dataflow. It is a high-throughput, low-latency data-parallel framework that supports iterative and incremental computations [Cho+16]. Differential dataflow models a program as a data-parallel dataflow over collections of records. Each record in differential dataflow is associated with a logical timestamp that denotes the point in time during program execution the record was produced. It includes operators such as map, join, group and filter, and also an iterate operator that deals with iterative computations.

[Cho+16] provide two main research contributions that address the two drawbacks of naive backward tracing. First, they provide methods to reduce the size of the explanations because too much information is difficult to process. Second, they provide a method for identifying explanations that would be sufficient enough to reproduce the result. Theirs is the only solution that provides complete output reproduction even with the presence of multiple non-monotonic operators in the dataflow. The framework creates a copy of the dataflow graph with its direction reversed, with its inputs being the outputs of the original graph and the unknown outputs that have to be discovered are the inputs of the original graph. The explanation of an output record is the set of input records that produced it. The framework indexes input records for every operator and saves them as outputs of the inverted copy of dataflow graph. However, it provides optimizations on join, distinct, reduce, iterate and top-k operators that does not require indexing of all of their input records.

Specifically, in the case of optimization of reduce operator, time constraints are considered. The framework can restrict inputs of an operator to only those with logical timestamp less-or-equal to that of the required output. A filter condition is added into the reduce operator as a rule to filter out inputs that do not satisfy this condition based on timestamp. Filtering of such records substantially reduces the volume of records in the reported provenance. This strategy that significantly reduces the number of records in explanations is the one which directly addresses the problem of too much information in naive backward tracking.

As stated earlier, in general data-parallel computations, the input records identified by naive backward tracing may not be sufficient to completely reproduce the output. The authors describe that this is due to the intermediate records that appear in computation of original dataflow graph but are not seen in its inverted copy. These records may interfere in downstream computation and change the results of operators, suppressing important outputs [Cho+16]. They state that this problem does not occur in dataflows of monotonic operators, as they have the property that additional input records only lead to additional output records.

The way to identify the records that are not part of reference computation is by using their property - that to affect any output, they must intersect the backward trace. That is, they execute the second copy of the program only on the inputs obtained from backward trace and the new records produced are these intermediate records. These records can themselves be traced backwards, and ultimately corrected through the introduction of more input records. Each new record thus intersecting the backwards trace is added to the backwards trace and traced backwards itself, possibly resulting in more required inputs, and this is repeated until convergence [Cho+16]. When the fixed-point is reached, the set of explanatory input records collected by the iterative backward tracing is sufficient to reproduce the given output.

The author performs evaluation of new backward tracing using the framework on the problem of graph connectivity to show how we can efficiently reduce the size of explanations using the techniques mentioned above. They also show that although explanations require many iterations of tracing in this case, they are still largely concise and interactive. The overhead incurred by provenance tracking ranges between 88% and 188%. They also evaluate the generalized backward tracing that provides explanations sufficient to reproduce the output and showed that the overhead incurred in providing “complete” explanations ranges between 316% and 358%.

2.4.6 Discussion

Maintaining and presenting full and exact provenance for dataflows in DISC systems may be infeasible due to its size and complex structure. And on top of that, we are using eager approach for collection which has size and performance overhead in itself. The survey on provenance tracking frameworks in DISC systems presented above is summarized in Table 2.1. All the solutions capture fine-grained provenance using the eager approach. RAMP and Newt do not provide access to intermediate data during process execution, whereas Lipstick, Titian and Explaining outputs [Cho+16] support interactive debugging allowing access to intermediate data as well. Though they provide sophisticated debugging and troubleshooting, the overhead incurred in terms of space is high. The space overhead tracking provenance of workflows in DISC systems is between 19% and 120% of the size of input dataset. In these systems, the data processed will be at the scale of gigabytes and terabytes, and hence provenance size of this scale is impractical and is not suitable for prolonged time in production systems. In the next section, we will discuss the solutions which aim to reduce the size of provenance.

2.5 Provenance approximation or summarization solutions

Provenance plays a noticeable role in query processing, debugging, data quality and understanding the underlying data. Capturing and storing complete or full provenance which describes every tuple in a result set and its derivation is prohibitively expensive. The previous section described that provenance data size grows too large when used with large data sets. Specifically, the size of provenance data grows up to 50% and 120% of size of input data in Titian[Int+15] and Newt[LDY13] respectively. Approaches to approximate or summarize provenance address

	RAMP [IPW11]	NEWT [LDY13]	LIPSTICK [Ams+11]	TITIAN [Int+15]	Explaining outputs [Cho+16]
DISC System	Apache Hadoop	Apache Hadoop, Hyracks, Apache Spark*	Apache Pig	Apache Spark	Differential dataflow
Granularity	Fine-grained	Fine-grained	Coarse- and Fine-grained	Fine-grained	Fine-grained
Eager/Lazy	Eager	Eager	Eager	Eager	Eager
View Intermediate Data	No	No	Yes	Yes	Yes
Size of evaluation data	100GB - 500GB	500MB - 1TB*	-	500MB - 500GB	-
Runtime overhead	20%-76%	10%-51%	upto 35%	30%-67%	88%-358%
Space overhead	19%-21%	30%-120%	-	30%-50%	-

* Newt applied to Spark in [Int+15].

Table 2.1: Comparison of provenance tracking frameworks in DISC systems

this challenge. In this section, we discuss the solutions that perform approximation, compression, or summarization on provenance data to reduce its overall size.

2.5.1 Approximated provenance for complex applications

[Ain+14] provides an algorithm that summarizes provenance expressions produced at the possible cost of information loss. The algorithm proposed in this work operates on provenance expressions captured and applies summarization rules on top of them. The summarization obtained helps to see trends and patterns in data rather than giving information on individual data objects. The algorithm takes into consideration the meaning of the underlying data (e.g. nationality of user, gender, or age) and then tries to merge or group annotations of similar data items based on it.

[Ain+14] uses certain abstract variables to identify individual units of data. These variables are provenance annotations. A certain finite set "Ann" of provenance annotations is fixed. To understand what data manipulation a certain data unit has encountered, the provenance annotations are combined with commutative semirings. The semiring annotations look as follows - $\cdot$ denotes the joint use of data, say as in join operations. $+$ refers to alternative use of data, as in union and projection operations. $\otimes$ denotes pairing of provenance annotation tokens

with data values for aggregation and $\oplus$ refers to aggregate function or combining multiple data terms. 1 and 0 tell if the data is present or absent. Conditional tokens like >,<,== resolve to True/False. An example of such provenance aware expression is $(U_1 \cdot A_1) \cdot [S_1 \cdot U_1 \otimes 5 > 2] \otimes 7 \oplus (U_2 \cdot C_2) \cdot [S_2 \cdot U_2 \otimes 3 > 2] \otimes 6$, where U_1, U_2, S_1 and S_2 are provenance annotations for individual data units and $\otimes, \oplus, \cdot$ are operators part of the semiring model.

This type of provenance aware expression provides all the information on every transformation undergone by the data units involved. This leads to a very long and complex expression. The algorithm presented by [Ain+14] summarizes such an expression to create a simpler and shorter version of it. A new set of annotations "Ann"' is now considered that corresponds to annotation summaries. The size of this new set Ann' is much less than the size of the provenance annotation set Ann. It then defines a mapping from every annotation in set Ann to its summary which is a summary annotation belonging to set Ann'. This mapping is a homomorphism, i.e, a function that assigns certain output value, given a particular input. Applying this homomorphism h to every annotation in the provenance aware expression, we get its corresponding summary expression. However, this summary of the real provenance loses track of some exact annotations and summarizes the provenance using the abstract annotations of set Ann'. For example, consider a homomorphism that maps all U_i and S_i annotations to 1, all A_i annotations to A and all C_i annotations to C (so Ann' = C, A). This homomorphism when applied to the provenance aware expression above produces the following simplified expression - $(1 \cdot A) \cdot [1 \cdot 1 \otimes 5 > 2] \otimes 7 \oplus (1 \cdot C_2) \cdot [1 \cdot 1 \otimes 3 > 2] \otimes 6$ which is equivalent to $A \otimes 7 \oplus C \otimes 6$, which is much more simple.

However, there may be multiple ways to define a mapping h, i.e., a provenance annotation can be mapped to multiple summary annotations present in set Ann'. Finding a good mapping is the main challenge of the algorithm. To define what a good mapping is, the algorithm considers the following factors - size of the provenance expression achieved by a particular mapping, the semantic constraints and distance. The distance factor quantifies the difference between the original provenance expressions and the summary expression. Semantic constraints allow only related annotations to be grouped together, that is, they impose certain constraints on which annotations from set Ann can be mapped to summary annotations from set Ann'.

To achieve the final provenance summary expression, the solution performs two main steps. First, it searches for annotations that are similar to each other and then combines them to produce multiple candidate summary expressions. Second, it measures the quality of these candidate summary expressions based on the three factors mentioned above and outputs the summary expression with best quality as the final result.

The task of finding similar annotations is based on computing equivalence classes in underlying data. Equivalence class is a set of elements where each element is considered to be similar to every other element in the class. The intuition behind combining annotations of data tuples in same equivalence class is that there is no need to maintain different annotations for each one of them, since they may not be differentiated. The algorithm first finds the set of equivalence

classes of provenance annotations in set Ann. It uses the idea of valuations for this purpose. Valuations are functions that assign true or false to annotations in set Ann. There can be multiple valuations which are part of the set V_{Ann}. Two annotations a_1 and a_2 can be considered to be in same equivalence class if and only if for each valuation v in V_{Ann}, v(a_1)=v(a_2). That is, two annotations have the same truth value for every valuation present in set V_{Ann}.

Thus, the algorithm computes equivalence classes from annotations with respect to a set of valuations. It examines two annotations at a time and considers them as equivalent to each other if they agree for every valuation in the set. The annotations in every equivalent class are then replaced by summary annotations using multiple possible mappings to produce the candidate summary expressions. In addition to obtaining a summary expression with small distance, the algorithm tries to minimize the provenance expression size, too. The distance and size measurements are combined together to form a weighted average[Ain+14] which is then used to find the best possible summary expression. This summary expression is then used for provenance-query processing instead of individual provenance expressions.

The experiments performed conducted with this solution aims at comparing their approach of grouping annotations based on equivalence classes to other approaches - Clustering and Random. Different experiments performed using different provenance datasets concluded that the given approach is indeed more fitting for the goal of finding quality summaries. In addition, the approach allows the user to control the desired tradeoff between distance (that affects evaluation accuracy) and size (that affects presentation and usage time) of the summary expression that can be obtained.

2.5.2 A provenance framework for data-dependent process analysis

Data dependent processes (DDP) model applications whose control flow is based on the state of underlying database and a finite state machine (FSM) [DMT14]. The work in [DMT14] provides a framework that allows static analysis of DDP executions using provenance. It uses the semiring based provenance model to interactively analyse the effect of any hypothetical changes performed on DDP's FSM or the database. This technique is called provisioning.

Consider a DDP shown in Figure 2.3. Figure 2.4 shows the underlying database and Figure 2.5 shows the SQL queries executed as part of this process. DDP shows the logical flow of the application and its dependency on data present in the underlying database. Each node is a web page. The initial node is Home page. The transition from one node to another induces some cost. For example, from Home page to New products, the cost is 2. Cost can be the time for execution or user effort required for this transition. Sometimes the transition is based on user decisions, as in the case of - Home page to New products, or on the state of underlying database, as in the case of the transition from Cat.(category) to Sub Cat.(sub-category). This transition is based on the execution of query Q_1. The query checks if there are any sub-categories available. If the query result set is not equal to 0, the transition is from Cat to Sub Cat, otherwise from Cat to Product.

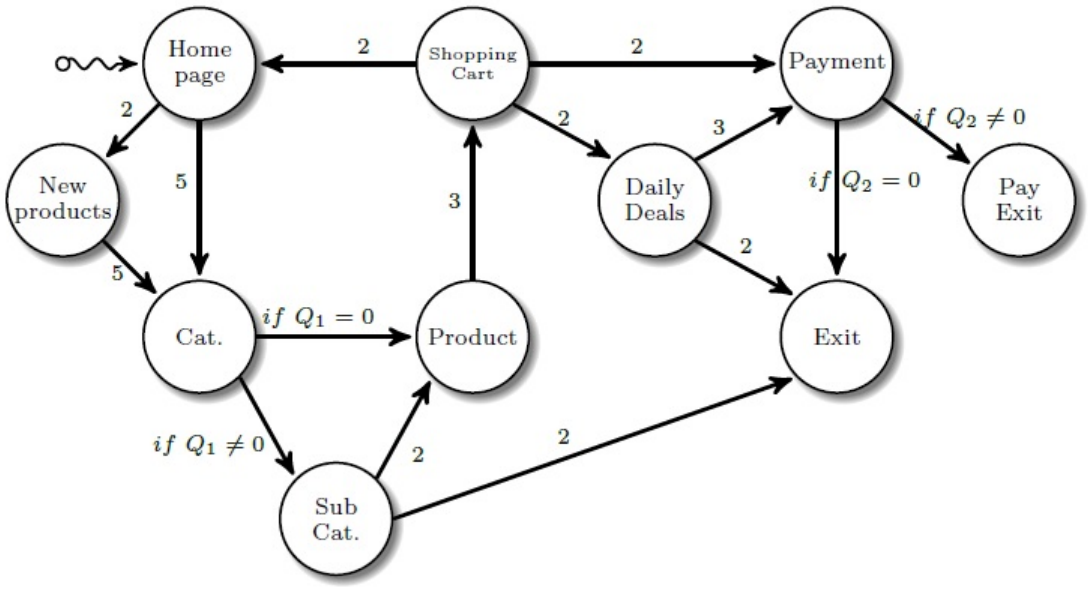


Figure 2.3: A Data Driven Process (DDP) [DMT14]

AvailableCat	
Cat.	**Prov.**
...	...
Cell Phones	d_1
Computers	d_2
Fashion	d_3
...	...

PaySys	
Cat.	**Prov.**
...	...
PayPal	d_5
...	...

CategoryHierarchy		
Cat	**SubCat.**	**Prov.**
...	...	...
Cell Phones	Smartphones	d_4
...	...	...

Figure 2.4: Underlying database [DMT14]

Analysts use Linear temporal Logic (LTL) [MP12] to denote their analysis tasks in provisioning. LTL contains –

1. Predicates that must be evaluated looking at the process states
2. Logical operators like and, or, not
3. Temporal operators that describe the required relationships between truth values of predicates during an execution.

For example, an analyst wants to analyse "A user exiting without viewing the daily deals" in the DDP shown in Figure 2.3. The LTL formula to express the scenario is (Exit $\wedge$ PayExit) B DailyDeals, where Exit, PayExit and DailyDeals are states, $\wedge$ is the and operator and B stands for Before, which is a temporal operator. Another temporal operator is F-finally, that states that the state after F holds true eventually. Example, the LTL formula for "A user views product sub-categories for some category" is F Sub Cat.

Q_1 :

```
SELECT COUNT(*)
FROM CategoryHierarcy CH,
     AvailableCat AC
WHERE CH.Cat = AC.Cat
```

Q_2 :

```
SELECT COUNT (*)
FROM PaySys PS
```

Figure 2.5: SQL queries ran as part of execution of DDP in Figure 2.3 [DMT14]

Semiring model of provenance is employed to collect provenance for result of these LTL formulae. We have seen semiring based annotations in Section 2.5.1. However, it is modified to accommodate the two kinds of transition choices – queries or user transitions. A first semiring will be used to annotate the database tuples and a second semiring will be used for annotation of user choices.

Consider the LTL formula F (DailyDeals $\wedge$ F Payment) and DDP of Figure 2.3. The framework constructs a provisioned expression for this LTL using semiring provenance model. The provisioned expression for an LTL formula contains multiple partial executions within it. The framework annotates every partial execution and multiplies multiple partial executions with each other to create one full execution that is represented by the LTL formula. The novel provenance model for every partial execution consists of two parts. The first part will specify the cost of the partial execution, and the second part denotes the dependency on data in underlying database. Let K and L be these two semirings respectively. If there is no dependency on data, the framework specifies the second part in annotation with 1. K $\otimes$ L will be the pair that captures provenance for each execution. Bag (K $\otimes$ L) represents the set of infinitely many executions that are possible to realize the LTL formula.

In case there are multiple ways to achieve a process state, sum of pairs are used to represent the many ways, where each pair captures the provenance of a single way. For instance, the two simple partial executions reaching Cat have a joint provenance which is the sum $\langle$ 5, 1 $\rangle$ + $\langle$ 7, 1 $\rangle$. The 1 in the second part of both annotations shows that there is no dependency on data. Also, there is a need for a mechanism to model and accommodate loops in DDP within the provisioned expression. For example, the transition from HomePage to Cat to Product to ShoppingCart. In this case, a new operator called Kleene star is introduced and applied to provenance of sub-executions appearing in a loop.

The provisioned expression obtained following the approach mentioned above is long and complex; it can be simplified using the following axiom based on the structure of the annotations. The framework applies the definition of bag multiplication to simplify the expression with bag semantics by performing point-wise multiplication. Point wise multiplication is applied to each pair of annotation that represents a single partial expression. Also, multiplication operation in the tropical semiring corresponds to natural number additions [DMT14].

Even after the application of the axiom above, the provisioned expression still is too large and its size depends on the size of the database. Congruence relations are now used to simplify this large provisioned expression. If the same data provenance is used repeatedly in multiple execution paths, the framework writes an equivalent expression for the same where it occurs only once. Congruence relation helps to identify elements of Bag (K x L) with “simpler” elements.

The framework also applies semiring homomorphisms to apply truth values to data tuples within the expression to simplify the provisioned expression further. Semiring homomorphisms can be considered as functions that assign a certain output value to a given input value.

Thus, the framework uses congruence relations and equivalence axioms repeatedly and finally applies semiring homomorphisms to obtain best possible simplified provisioned expression. The size of this expression is dependent only on the number of states of the FSM and number of parameters, in contrast to non-simplified provisioned expression which depended on the database size.

Evaluation performed as part of this work aimed at assessing three factors - size of the provisioned expression obtained, time it takes to generate this expression and the time it takes to use this expression for analysing results under hypothetical scenarios. Provisioned expressions obtained are close to 180MB for a database with 5 million tuples. The authors also examine the effect of the size of FSM on the size of expression. Although theoretically this is exponentially bound, when simple structures(topology) of FSMs are used, this exponential bound is not met. It is observed that FSMs with simple topologies produce simple expressions. The time required to obtain the provisioned expression is close to 35 seconds when the database contains 5 million tuples. The experiments prove that the process is faster when serial and parallel finite state machine topologies are used, however it is significantly slower for complex dense structures which have about 5000 process states.

2.5.3 Provenance distillations

Scientist must curate the dataset produced by their experiments by adding metadata on its derivation process, sources and owners. The work in [Alp+13] proposes using provenance traces generated during workflow execution to assist scientists in data curation. They provide a way to distil provenance information by extracting only the important and interesting information from within detailed provenance traces. This interesting information necessary for curation contains summary of the workflow and details of sources that created the result dataset. Provenance traces cannot be used as is in curation because of the following limitations

1. Provenance obfuscation - No differentiation between important and non-important metadata.
2. Lineage Opacity - No clear relationship between input data and output data after every transformation is specified.

3. Coarse Granularity - Assumes that every output data item depends on all of input data items.

Distilled provenance is necessary in this regard. This distilled lineage contains succinct origin annotations on result data artifacts and highlights of significant activities of the workflow. The origin annotations describe how data originated and its derivation history. To distill provenance, the initial step is design time annotation of activities within workflow descriptions. These annotations categorize the data oriented activities such as data retrieval, data analysis etc. These categorizations are called motifs. Motifs characterize the data oriented nature of activities undertaken by workflows. These motifs are necessary for both runtime origin annotation generation and workflow summarization.

These motif annotations can be applied manually or semi automatically. The semi-automatic annotation can be performed using a classifier that mines existing workflows to suggest motif annotations for the current workflow. Based on these motif annotations, only the activities and their annotations based on the requirements of the scientist are extracted. Hence, Motif annotations could be used to generate workflow summarizations by eliminating the secondary or non-important steps and retaining only the important ones in the workflow.

The authors of this work provide a motivation to distill provenance to support scientists in curation of their experimental results. However, they have not performed evaluation to assess the effectiveness of this form of distilled provenance as of now.

2.5.4 Ariadne: Managing fine-grained provenance on data streams

[Gla+13] provides a fine-grained provenance tracking framework for data stream management systems (DSMS) and performs compression techniques on the collected provenance. These compression techniques are based on the underlying structure of data streams and aim to reduce size of provenance that leads to faster provenance-query processing. Tracking provenance in data streams has challenges that include infinite data arrival, low latency provenance tracking etc. However data streams have a characteristic that favours compression. They usually have an ordered data model, and hence ordering can be exploited in compression of provenance of data streams.

Ariadne tracks provenance by annotation of tuples in data streams and operator instrumentation. Operator instrumentation involves modification of DSMS operators to capture and propagate provenance annotations. Every tuple in a data stream is represented by a tuple-identifier (TID) that uniquely identifies the tuple. Operator instrumentation is performed based on these TIDs. Provenance is represented as a set of contributing tuples from input or intermediate streams. Each tuple is annotated by this provenance set that describes the derivation of this tuple from its sources. For example, if tuple “D” with TID 4 is based on tuples “A”, “B”, “C” with TIDs 1,2,3 respectively, then the provenance for tuple “D” would be the TID-set 1,2,3. These

provenance TID-sets contain individual TIDs which are ordered and this property is exploited by the compression methods described in following paragraph.

Ariadne uses compression methods that are window based, as aggregation on data streams is usually performed on certain window of data. A window in a data stream is a certain finite subset of tuples present in that stream. Ariadne presents the following compression methods.

Interval Encoding Interval Encoding utilizes the fact that aggregation in data streams is windowed and that the provenance of window of data is the provenance of union of individual data tuples present in the window. The TID-set is a continuous sequence and when there is aggregation required, the window consists of TID-set of continuous sub-sequences. Interval encoding works on these sub-sequences. If the provenance of a result tuple of an aggregation operation is the TID-set [1,2,3,4,5], interval encoding compresses it as [1,5]. Interval encoding works well if the sub-sequences consist of many continuous tuples. However, it incurs overhead if there is no aggregation as every tuple needs to be represented by both the start and end of TID-set interval.

Delta Encoding Delta encoding exploits the characteristic that windows with small slide values overlap to a large extent. Hence, the TID-Set of a tuple can be represented as some delta to the TID-Set of one of its predecessors. A tuple with uncompressed provenance is sent succeeded by several tuples with their provenance encoded as a delta to the last uncompressed provenance that was sent. The last complete TID-set sent and the number of deltas applied to it must be stored. Hence a TID-Set can be restored from its delta representation in a single step without applying a multiple deltas to the last uncompressed provenance.

Dictionary Compression The TID-sets can also be encoded using dictionary compression methods like LZ77. However, they incur processing overhead of compressing and decompressing TID-sets before querying provenance.

Adaptive Combination of Compression Techniques The techniques mentioned above can be applied with each other based on heuristic rules. For example, the interval encoding is first applied. If sufficient overlap exists in TID-sets, delta encoding is then employed. If further compression is required, the dictionary methods are used. These compression techniques however can be applied conditionally, that is if there are operations that operate on long continuous TID-set sub-sequences or if there is sufficient overlap between TID-sets. The dictionary methods introduce runtime overhead and hence the user must trade runtime overhead for storage costs.

The evaluation performed as part of [Gla+13] compares the latency incurred due to provenance generation when compression techniques are used to when they are not used. Without any optimizations or compressions, the framework incurs a latency overhead of 75% for provenance generation. When adaptive combination of compression techniques is applied, the overhead reduces to 60%.

	PROX [Ain+14]	PROPOLIS [DMT14]	Provenance Distillations [Alp+13]	Ariadne [Gla+13]
Purpose for which provenance is reduced	Eager	Eager	Eager	Eager
Purpose for provenance reduction	-	-	Curation	-
Approach	Approximated summarization using equivalence classes based on provenance expressions	Approximation using congruence relations and equivalence axioms on provenance expressions captured	Summarization to store lineage for only "interesting" tasks	Compression of collected provenance using data stream-specific compression methods
Granularity of Captured Provenance	Fine-grained	Fine-grained	Coarse-grained	Fine-grained
Eager/Lazy	Eager	Eager	Eager	Eager
Provenance data reduction during execution	No	No	No	Yes
Provenance reduction rate	Eager	Eager	Eager	Eager

Table 2.2: Comparison of existing provenance approximation or summarization systems

2.5.5 Discussion

The solutions described above operate on full provenance and perform different techniques on them to produce partial or imprecise provenance. Partial provenance includes provenance for only a subset of rows of the input dataset, and only this reduced partial provenance is used for answering provenance queries. Even though there is loss of information in this case, a system may be able to tolerate this if a large amount of storage space is saved and provenance-query processing is faster. Partial provenance obtained by approximation/summarization/compression can provide an overview of underlying data. The summarized partial provenance can be mined to obtained contextual information related to the underlying data. For example, in a movie rating application, contextual information about users can be found based on aggregation or summarization. Insights like "Male users of age group between 20-25 provide rating of 5 stars for action movie" can be obtained.

Table 2.2 compares the solutions which optimize and reduce the amount of provenance data. PROX [Ain+14] and PROPOLIS [DMT14] are based on equivalence class based approximations. They work on full provenance data and approximate it after its collection. PROPOLIS also uses congruence relations to find the summarized provenance expression which can be produced from multiple provenance expressions of individual records. [Alp+13] introduces motif annotations to identify certain important tasks and provides summarized provenance based only on these tasks. Ariadne [Gla+13] uses compression techniques like delta encoding and interval encoding that exploit the ordering present within provenance data to provide compressed provenance. All these solutions deal with fine-grained provenance, except for provenance distillations whose main application is in curation. Thus capturing fine-grained provenance dealing with every data item in the input is irrelevant in this case. Each of these solutions reduce the amount of provenance data after the execution of the process. As these solutions collect provenance in the eager way, provenance is collected when the process is being executed. The amount of provenance collected is still huge as the optimization techniques discussed above do not apply during the execution. The size of the reduced provenance data will still be between 30% and 50% of the size of input dataset as discussed in Section 2.4. Thus it is too large to be used permanently in the production systems.

The aim of this thesis is to identify data value reduction techniques that reduce the amount of provenance data at the time of its collection. Before we dive into the data value reduction techniques, in Chapter 3, we introduce the applications of provenance. We see that none of the solutions discussed above, except for [Alp+13] have been tailored to a particular application of provenance. Provenance collection can be better optimized if the purposes of provenance collection are known in advance. Hence, it is important to identify the use cases and applications of provenance so as to develop application-dependent provenance data reduction techniques. Chapter 4 introduces such techniques and presents the applications they can support.

3 Applications of provenance

This chapter is dedicated to understanding the applications or use cases of provenance. In chapter 2, the solutions which approximate or summarize provenance data have been described. However, none of these solutions perform approximation at runtime when the workflow is executed. Also, none of the solutions except for [Alp+13] have been tailored to a particular use case of provenance. So if the purposes of provenance collection are known in advance, provenance collection can be optimized in a custom manner. Hence, this chapter presents the use cases of provenance with the aim to identify approximation techniques in the best possible way. The applications of provenance can be divided into the following categories as illustrated in Figure 3.1 - monitoring, auditing, debugging, data quality, data exploration, data security, data replication, attribution and context information. In the following sections, these categories and their respective sub categories are briefly discussed. However, this is a collection of use cases and applications of provenance in dataflow engines. Other categorization for a broader field of applications exists [HDBL17].

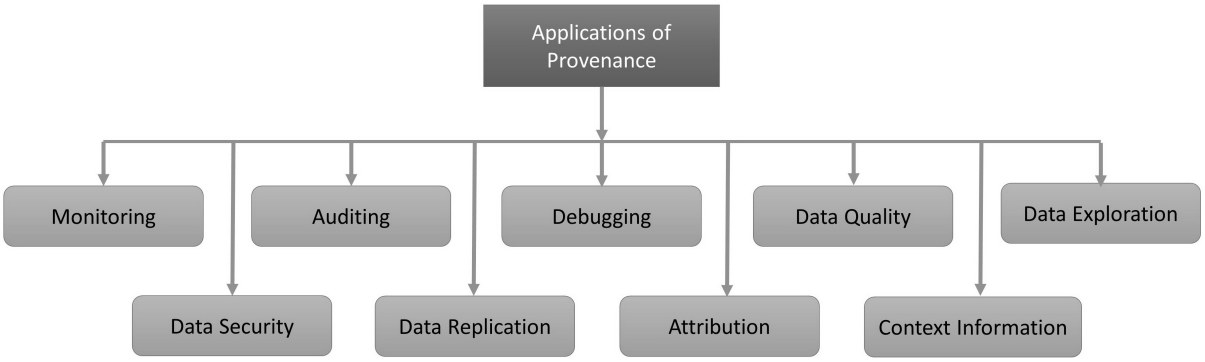


Figure 3.1: Applications of Provenance

3.1 Monitoring

Provenance can be used for observing the progress of a program execution. It can be used to monitor usage of system resources and maintain availability and functionality of the system. Provenance understands the underlying semantics of the data and hence can be a part of alert systems providing following alerts:

Row-level integrity alerts [OR11] These alerts notify the administrator or operator of the system in case a data tuple fails to satisfy a given condition (E.g. Column "Age" > 0).

Table-level integrity alerts [OR11] These alerts notify the administrator or operator of the system in case a set of intermediate records violate a given condition. (E.g. Cardinality > 0)

Latency alerts [OR11] These alerts notify the administrator or operator of the system when a certain transformation takes more time to process a certain data item than the time taken by a valid record. This data item can then be checked thoroughly to understand what caused the processing delay.

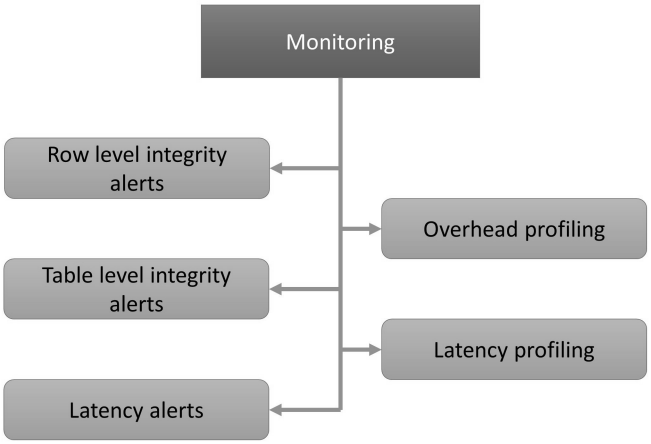


Figure 3.2: Applications of Provenance in Monitoring

Provenance can track information on time required by a record to process through a certain operator or through the entire process execution. Hence it can be used in profiling time overheads and record processing latencies.

Overhead profiling [OR11] presents the operator wise division of total time of process execution.

Latency profiling [OR11] describes the latency distribution of processing of data tuples across multiple executions with different data sets.

3.2 Auditing

Provenance can provide auditing for the data processes and the results they deliver. It tracks the flow of information within a process. The evaluation of this audit data can be used to understand the weakness in the process and help to optimize it.

Audit Trail [Gro+05] Provenance tracks all the information necessary to describe in detail what processes or data influenced a certain item of interest and can serve as an audit trail for that item.

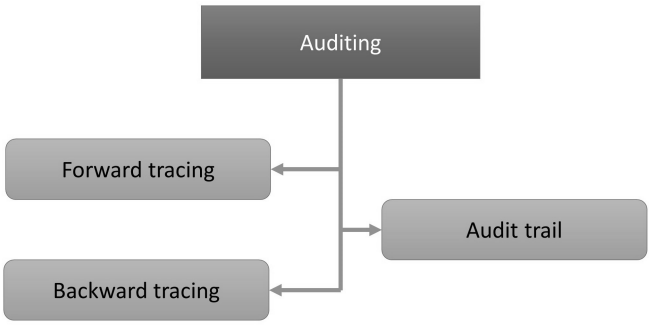


Figure 3.3: Applications of Provenance in Auditing

Backward and forward tracing [OR11] Provenance as a backward trace contains input and intermediate data corresponding to a particular output datum. Provenance in the form of forward traces finds the intermediate and output data tuples that arise from a given input data tuple. Hence provenance as traces of process execution can also be used in tracking missing records, investigating faulty data and demonstrating compliance with required standards.

3.3 Debugging

In today's world where usage of large sets of data for analysis is common, debugging of the processes which deal with such data is of utmost importance. Provenance data about a process or a workflow helps debugging it. It can be used to perform root cause analysis when an error occurs, find wrong or invalid data that caused failure or even in step wise visualization of how an end product evolved from its source.

Crash culprit determination [OR11] Provenance helps to identify the record or transformation as a culprit that initiated a failure.

Data summaries [OR11] A histogram of data records on a certain transformation within a process execution is recorded and stored as part of provenance data. It is then compared against

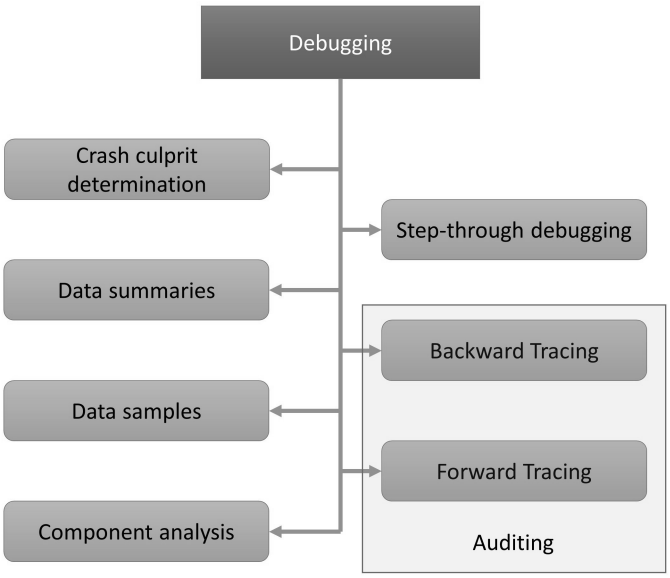


Figure 3.4: Applications of Provenance in Debugging

statistical summaries from previous process execution on the same operator. This helps to find data distribution changes which might have stemmed from a failure or an exception.

Data samples [OR11] involves passing a small sample of data tuples across every operator in the process as a sanity check. Provenance is stored for this sample of data and is investigated to check if there is any invalid data that might cause a failure within the process.

Step-through debugging [OR11] allows programmers to step through the code and view data obtained after every operator execution. Provenance contains details on the operator's inputs and outputs and hence we can view them to debug an error along the execution.

Component Analysis [OR11] Using provenance, we can determine what components of the dataflow were involved in generating a particular output data tuple.

Backward and forward tracing can be used in debugging in addition to Auditing.

Backward Tracing Backward trace of a record will describe the manipulations undergone by it from its source record(s). In the event of an error, backward tracing helps programmers find faulty components or invalid data that caused the error and apply relevant corrections.

Forward Tracing Provenance as forward trace finds the intermediate and output data tuples that arise from a given input data tuple.

3.4 Data quality

Data are abundantly available for public use due to increase in number of data portals and easily accessible cloud-based data markets. The sources of this data exhibit heterogeneity in their quality. Hence, not all sources can be considered reliable and trustworthy. Provenance helps understand the origin of the source of an end product, hence allowing for estimation of its quality.

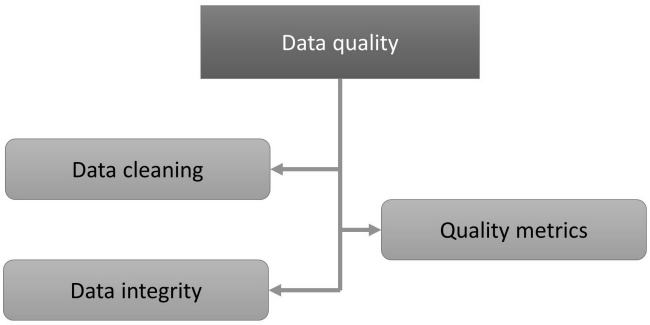


Figure 3.5: Applications of Provenance in Data Quality

Data Cleaning [Gal+01] The erroneous data which was included as part of the execution due to the "unclean" sources can be detected by provenance and removed. Hence provenance can support data cleaning.

Quality metrics [JO04] If Provenance holds sufficiently detailed information on sources and semantics of underlying data, it can be used to compute data quality metrics, and a trust score can be presented for each of the sources and its corresponding data.

Data Integrity is based on the trustworthiness of sources used and intermediate components that access the data during dataflow. As provenance provides information on what sources are included as part of the data and from where these sources have been extracted, data integrity issues can be addressed using it.

3.5 Data exploration

The provenance data can be queried and used in data exploration, to provide insight on the underlying data and the data derivation process. Interesting patterns can be discovered from multiple sources by mining the provenance data. Provenance can also be browsed in graphical forms like tree for better visualization of dependencies within data. It helps in identifying the effect of modifying certain data set or a sub task on the execution of entire process and its output.

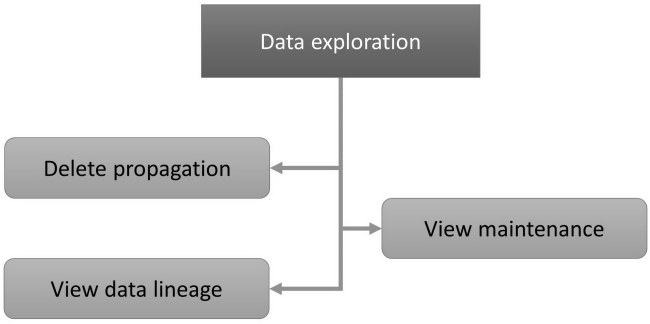


Figure 3.6: Applications of Provenance in Data Exploration

Deletion propagation involves analyzing how potential deletions propagate through the workflow execution, allowing users to assess the effect that tuple t has on the generation of some other tuple t' [Ams+11]. As provenance knows the dependencies between the tuples, it can be used in deletion propagation. Hence, it helps in understanding how potential deletions will impact the end result as well as the intermediate data.

Provenance is used in maintenance of views. A view is a virtual table based on the result set of query executed on certain source data. When source data is changed, the view based on it has to be changed accordingly. Provenance data in this case helps to find the view associated with the modified source data so that it can be modified accordingly.

View data lineage provides a set of tuples in source tables which led to a certain tuple present in the view.

View maintenance Provenance contains information on what source tables are associated with a particular view, and whenever a source associated with that view changes, the view is changed to reflect new changes. For example, it helps to find tuples in a view, which have to be deleted after their corresponding tuples have been deleted from their respective sources.

3.6 Data security

Provenance and security can be considered to be closely related. As data provenance describes the history of the data from its origin to its current state, it can be used to establish privacy and better access control to data. It can also be used to confirm the adherence of process or its data to regulatory compliance rules. Provenance ensures that a user's sensitive financial information is not leaked to other users using an information flow policy [CSH07].

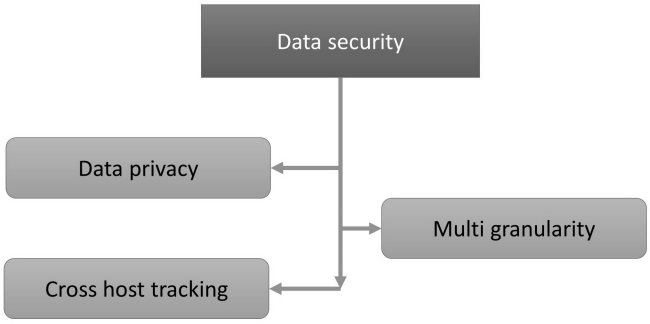


Figure 3.7: Applications of Provenance in Data Security

Data privacy Based on the provenance annotations or data, decide which user will be allowed to access what data in the system.

Cross host tracking [TKH13] involves tracking hosts across distributed environment which accessed the data. This facilitates understanding the flow of data across distributed network. Data is transferred across multiple machines and geographies and hence, it is mandatory that only trustful objects are allowed to access to the data. Provenance helps in this area by permitting only valid or 'good' components and machines to access the data and ignore the rest.

Multi granularity [TKH13] Collecting provenance for components and data exchanged between different levels of architecture will help the data analyst understand working of the entire system. This will enable him control access to data from end-to-end, and detect intrusions from their inception and overcome them.

3.7 Data replication

The end product of a process can be computed again based on its provenance data as it includes in detail the steps involved in deriving it from its sources. Exact replication of results may be performed when detailed information on the transformations, operations, parameters and sources is present in the provenance data [Fos+03]. Also, repeatability of results requires that the similar environment be present as was when the process first ran to produce the result. Repeatability is necessary when the source data or the intermediate tasks within the process are modified, and we still need the older results. It is sometimes cost effective and hence preferred over transporting, copying or storing of the result data.

3.8 Attribution

The ownership of source data can be confirmed by its provenance. Users of this data can look at its provenance on how the data is derived. This also helps to ascertain the identity of its creators and establish its copyright [JO04]. The creators of certain data can look at the provenance and see how and where their data is being used. Provenance can be treated as some form of citations in this context. Also it can be used to assign responsibility to the right owner in case there is a bug in the dataset.

3.9 Context information

Provenance contains metadata or information which can be helpful for data discovery. Using this metadata, searches can be performed based on the sources used in generating this data or also based on the tasks or transformations executed on it. These searches can eliminate the task of rerunning certain process when we already have the results for it. Provenance data usually contains annotations with it which help associate the data with its domain. Hence the user can relate to the right context the data belongs to by examining only the annotations and not the entire dataset. This task is especially of interest for archived data that was generated long ago.

3.10 Discussion

The use cases and applications of provenance described above pertain to provenance in dataflow engines. Many of these applications do not require fine grained provenance of every tuple in the input. Such applications that do not necessarily need provenance for every tuple provide potential for data reduction. Applications like data security, data replication etc., usually require fine grained data provenance related to all the input data. However, applications like latency profiling, data quality etc., do not require the derivation process for all of its input. These applications are called data-agnostic applications. In the next chapter, we introduce reduction techniques that can be applied to data provenance which is used in data agnostic applications.

4 Provenance data reduction techniques

The previous chapter described the use cases and applications of provenance like data exploration, debugging etc. This chapter presents the data value reduction techniques that can be applied to provenance during its collection. These data reduction techniques can be better optimized based on the use cases and applications of provenance data. The techniques described in the following sections collect provenance for a subset of input data instead of all of them. That is, they collect partial provenance. Partial provenance requires less space compared to full provenance and supports fast processing of queries on provenance data. We collect partial provenance at a possible cost of information loss. A consequence of using partial provenance is that the result set of a query executed on this provenance may differ from its true value. This difference will be different for different data reduction techniques. We present few use cases and applications for provenance which may be able to tolerate this difference and apply the corresponding data reduction technique to them. This chapter describes in detail the following data reduction techniques which can be applied to provenance - sampling, histogram analysis, clustering and equivalence classes. These techniques will be evaluated on top of Apache Spark. We also provide an outlook on the following techniques - Deduplication, Outlier detection, Stream Summary, Count-Min Sketch and Locality Sensitive Hashing and Min hash towards the end of this chapter. Evaluation of these techniques is not part of this thesis and can be performed as future work.

4.1 Running example

Provenance describes the origin and the evolution of data along its lifecycle. Provenance of a tuple t present in the result of a program P executed on certain data D is the set of tuples in the input that contributed to or helped produce t. Example data is shown in Table 4.1. We execute Spark program P on this example data to obtain average temperature per country. The following is the program P.

1. $data = sc.textFile(\text{"}Temperatures.csv\text{"}, 1)$
2. $countryTemperature = data.map(p => p.split(\text{", "})(4), p.split(\text{", "})(3))$
3. $temp1 = countryTemperature.mapValues(x => (x, 1))$
4. $temp2 = temp1.reduceByKey((x, y) => (x._1 + y._1, x._2 + y._2))$
5. $averageTemperaturePerCountry = temp2.mapValues(x => (x._1/x._2))$
6. $averageTemperaturePerCountry.collect$

	Date	Latitude	Longitude	Temperature	Country
t_0	26-04-2016	66	-153	51	Alaska USA
t_1	26-04-2016	68	-151	39	Alaska USA
t_2	26-04-2016	65	-152	60	Alaska USA
t_3	26-04-2016	-16	145	18	Australia
t_4	26-04-2016	-16	140	20	Australia
t_5	26-04-2016	66	-153	54	Alaska USA
t_6	26-04-2016	65	-151	49	Alaska USA
t_7	26-04-2016	65	-152	54	Alaska USA
t_8	26-04-2016	-33	151	23	Australia
t_9	26-04-2016	-33	158	17	Australia
t_{10}	26-04-2016	-31	115	18	Australia
t_{11}	26-04-2016	-31	118	23	Australia

Table 4.1: Example data: Temperatures at different locations

t_a	Alaska USA	51.66
t_b	Australia	19.83

Table 4.2: Result of program P executed on data in Table 4.1

Statement 1 in the program loads the data from file specified and assigns the content to the reference - data. Statement 2 extracts country and temperature from each line and pairs country name with temperature and assigns these pairs to reference - countryTemperature. Statement 3 and 4 compute the sum of temperatures for each country and the number of occurrences of each country and assigns them to reference - temp2. Statement 5 computes the average of temperature for every country and assigns the result to reference - averageTemperaturePerCountry. The collect action in statement 6 triggers the evaluation of the averageTemperaturePerCountry reference, and all transformations leading up to it. Table 4.2 illustrates the result of program P. This result corresponds to non reduced data, and full provenance is collected for all the tuples, i.e., t_a and t_b present in this result.

For instance, provenance of the tuple t_a in the result of program P are the tuples t_0,t_1,t_2,t_5,t_6 and t_7. These are the tuples from the input that contributed to the result tuple t_a. The program has application in Data exploration, where an analyst may want to know the weather conditions of certain locations at different times of the year. One of the provenance queries that can be executed on these tuples is - Why is the average temperature in Alaska USA greater than that of Australia? The tuples belong to the month of April and it is not normal that Australia is colder than Alaska in summer. Inspecting provenance tuples and looking at the attribute “Temperature”, we notice that the values range between 49 and 60. The number of countries that record temperatures in Fahrenheit are limited and the data source does not contain any information on units. Hence, by looking at the results, it is not clear that some values are provided in Fahrenheit

and the analyst might consider all the temperatures would be in Celsius if she does not inspect provenance data thoroughly.

In this example, we have 6 tuples that correspond to provenance of the tuple t_a. However, in DISC systems, we deal with millions and billions of tuples and the number of tuples that contribute to the presence of a certain tuple in result of a query can be in millions or billions. These tuples correspond to the size of the provenance stored associated with result. For example, if we run the program P on data that contains 5 million tuples from Alaska USA, then all these 5 million tuples contribute to the result tuple t_a. The provenance for result tuple t_a will be based on all these tuples. We collect exact or full provenance in this case where we collect provenance for every tuple in the result and contains information on every tuple that contributed to every tuple present in the result. However, the size of the provenance data collected depends on the size of input data. We discussed in Section 2.4 that the size of full provenance collected by solutions that collect full provenance in DISC systems is approximately 50% of the size of input data. Hence, we look at ways to collect partial provenance such that we reduce the overall size collected and stored. In the following sections, we describe such techniques.

4.2 Sampling

"Data sampling is a statistical analysis technique used to select, manipulate and analyze a representative subset of data items in order to identify patterns and trends in the larger data set being examined" [Mar16]. It is a method of selecting a small subset of data from a large dataset. It is illustrated in Figure 4.1. Out of 12 items in the input, we select only 4 items. The size of this sample is 33% of that of the input data.

There are many reasons for using samples. Most often, the cost in time and effort prohibits gathering information from the entire input data. Importantly, with high-quality data collection methods and appropriate ways of selecting a sample, we can obtain accurate information about the input. Sampling helps data analysts to work on a limited quantity of data derived from the large input data to accelerate their analysis, while still producing near-accurate results. Sampling is particularly useful with data sets that are too expressive to efficiently analyse in full. For example, Sampling is used in election polls to predict results. It is impractical to poll every voter of the election, as that number would be in hundreds of millions. That is when sampling comes into picture. The polling agencies select only a sample of voters that represents the whole population. The results of the polls are predicted based on the votes given by this sample of voters.

The frameworks introduced in Section 2.4 that collect provenance in DISC systems like [Int+15] produce provenance whose size is comparable to the input data used in these systems. It is impractical to use provenance of such size to be used permanently in production environments. We can apply sampling to input data or to provenance data. In both cases, we will be collecting

provenance for only a sample of input data and not on all of them. Hence we reduce the size of provenance collected and stored by collecting partial provenance instead of full provenance.

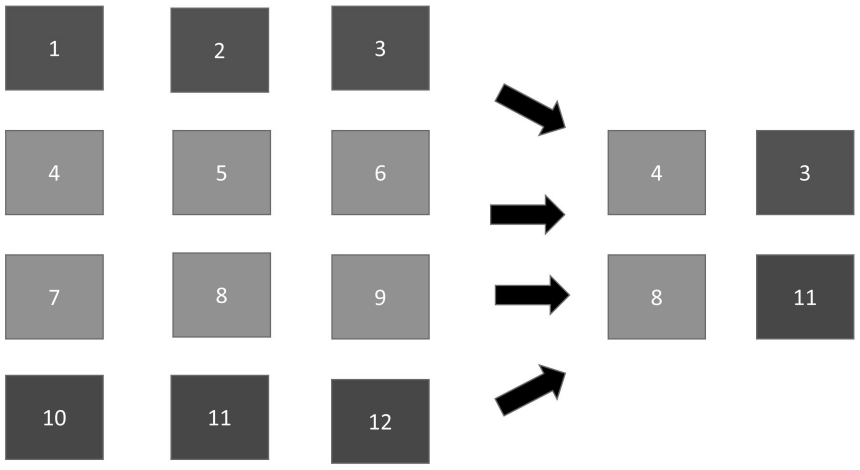


Figure 4.1: Data sampling

Best samples are the ones that are free from any bias. Every data item in the input data should have a more or less equal chance of being a part of the sample. If a sample has some data items from input data have less chance of being a part of it, then the resulting sample is a biased sample, and the results obtained from such a sample can not be valid or generalizable to the entire input data. The size of the sample is an important aspect of sampling, too. Sometimes, a sample of small size may tell most of the important information about input data set but this cannot always be guaranteed. A larger sample will of course be closer to the whole data set, however the increase in the size of the sample may impede ease of manipulation and interpretation[Mar16]. Either way, samples are best drawn from data sets that are as large and close to complete as possible.

In sampling, the term population refers to the non sampled input data. Sampling error is the difference obtained from calculating the estimate (estimated mean, total, proportion, etc.) based on a sample rather than the population. This is because the value estimated by the sample may not be exactly equal to the true value of the population. For example, in prediction of election results, the sample of voters considered might not always predict accurately what the entire population votes for. The measure used to estimate the sampling error is the standard error and is used to quantify the accuracy of the estimation.

There are two main approaches to Sampling - Nonprobability sampling and probability sampling.

1. Nonprobability sampling is a sampling method where some elements of the population have no chance of selection.
2. Probability sampling is a sampling method in which every unit in the population has a non-zero probability of being a part of the sample. This probability can be accurately determined.

We will focus our attention on the following probability sampling methods:

Simple random sampling

In a simple random sample of a given size, all data items present in the population are given an equal probability of getting selected into the random sample. Any given pair of items has the same chance of selection as any other such pair (and similarly for triples, and so on). There are two ways we can create a random sample - with replacement and without replacement. Figure 4.2 illustrates them. Let the population contain numbered square blocks. The sample without replacement as shown in Figure 4.2 a) may only contain distinct square blocks, whereas the sample with replacement as shown in Figure 4.2 a) may contain the same square block multiple times.

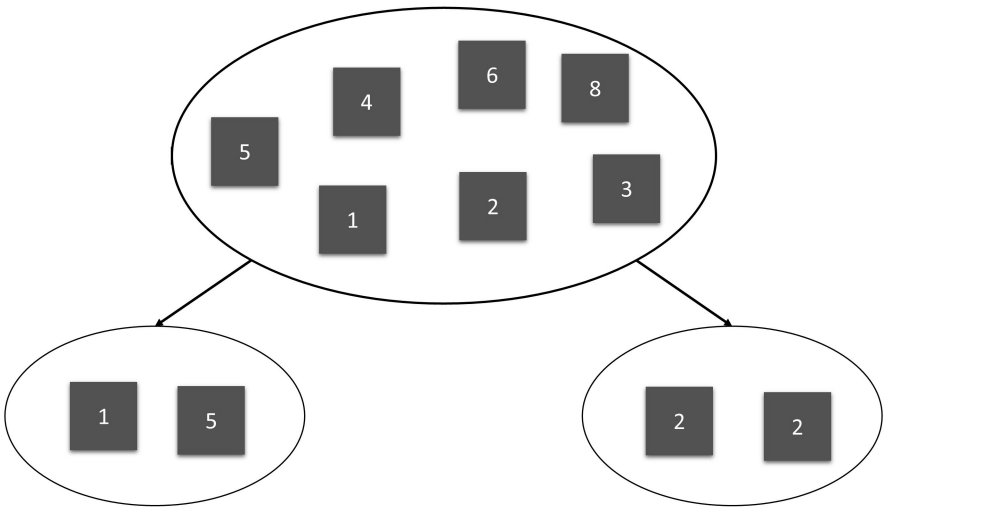


Figure 4.2: Simple random sampling

Consider an example where out of 1000 employees, a manager wants to select 100 for assigning a new task. 1000 names may be put in a bowl and 100 names can be selected from them. In this case, all the employees have equal probability of getting selected and this probability can be accurately determined.

The probability (P) of a person being selected can be calculated from the sample size and the size of the input data. P that any data item can be selected randomly without replacement (can be selected only once) is:

$$
\begin{aligned}
P &= 1 - \frac{N-1}{N} \cdot \frac{N-2}{N-1} \cdot \cdots \cdot \frac{N-n}{N-(n-1)} \\
&\overset{\text{Canceling}}{=} 1 - \frac{N-n}{N} \\
&= \frac{n}{N} \\
&= \frac{100}{1000} \\
&= 10\%
\end{aligned}
$$

P that any data item can be selected randomly with replacement (can be selected more than once) is:

$$P = 1 - \left(1 - \frac{1}{N}\right)^n = 1 - \left(\frac{999}{1000}\right)^{100} = 0.0952 \cdots \approx 9.5\%$$

Substituting the values for sample size and size of input data as 100 and 1000 respectively in equations above, we get 10% and 9.5% respectively. This means that every employee has around 1 in 10 chance of being assigned the task.

The basic rule used for estimating population parameters such as mean is that a population mean is estimated based on the corresponding mean in the sample [All97]. Also, in case of databases, queries can be executed on sample instead of the input data to obtain a faster and near accurate result. In this thesis, we are interested in reducing the size of provenance by creating sample from input data and collecting provenance only for that sample. In this case, collecting provenance for unique data items would provide us with most information on the input data than having same data item selected multiple times within a sample. Hence, we will be using simple random sampling without replacement in our evaluation in Chapter 5.

Stratified sampling

The population consists of different types of data items. It is advantageous to create samples based on these different types. Stratification is a method of separating data items of the population into uniform sub-groups before performing sampling on them. The sub-groups created are called as strata. The strata should be mutually exclusive. Every data item in the population should be a part of only one stratum. Each stratum is then sampled as an independent population, on which random sampling is performed in order to obtain the stratified sample.

Figure 4.3 illustrates stratified sampling. Consider a scenario where we must estimate average number of votes for each candidate in an election. Let the state has 3 villages: Village A shown as triangles has 1000 people, Village B shown as circles has 2000 people and Village C has 3000 people. We can create a random sample of size 60 over entire input data, however that might not be well balanced across these villages. Instead we perform stratified sampling. We first divide the state's population in three strata corresponding to each village. And then we perform simple random sampling on each of the strata based on the size of strata and the required size of the sample. Hence we choose a random sample of 10, 20 and 30 from Villages A, B and C respectively. These samples are combined to form the stratified sample of size 60 which is well balanced across every strata.

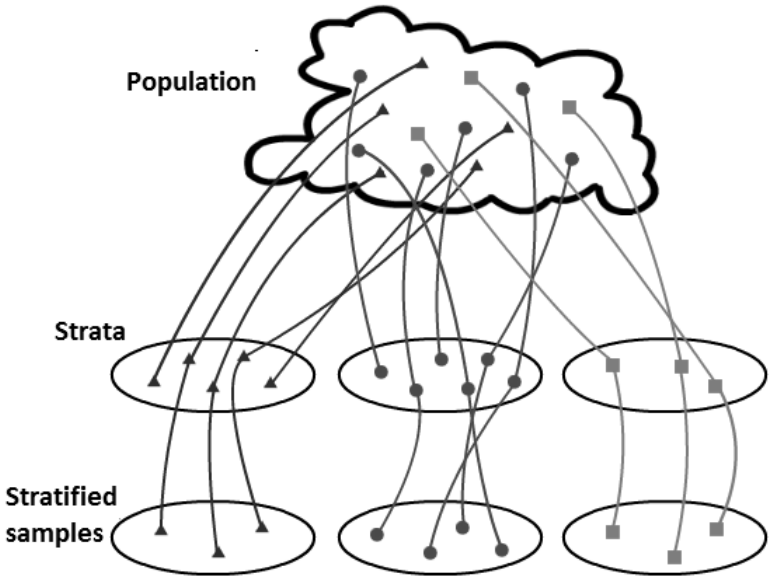


Figure 4.3: Stratified sampling [Vin14]

As in case of simple random sampling, the population parameters of the input data can be estimated based on the population parameters of the sample. For example, the mean of the population can be estimated based on the mean of the stratified sample.

	Date	Latitude	Longitude	Temperature	Country
t_0	26-04-2016	66	-153	51	Alaska USA
t_1	26-04-2016	68	-151	39	Alaska USA
t_2	26-04-2016	65	-152	60	Alaska USA
t_3	26-04-2016	-16	145	18	Australia
t_4	26-04-2016	-16	140	20	Australia
t_5	26-04-2016	-33	151	17	Australia
t_6	26-04-2016	-31	115	18	Australia
t_7	26-04-2016	-31	115	23	Australia

Table 4.3: Simple random sample of example data of Table 4.1

Sampling for provenance data reduction

We can use sampling as provenance data reduction technique to reduce the size of provenance collected. Consider the example data presented in Table 4.1. We apply simple random sampling with sampling ratio of 66% to obtain a subset of data smaller than example data given. Table 4.3 shows the random sample with 8 tuples obtained from the example data of 12 tuples. The random sample may contain any 8 tuples from the 12 tuples from the input as we have asked for sampling ratio of 66%. Now we execute the same Spark program P as in Section 4.1 on this random sample.

Provenance of the tuple t_a present in the result of the program P on data in Table 4.3 is now the set of tuples, t_0, t_1 and t_2. We see that the provenance now corresponds to only 3 tuples instead of 6, as was the case in Section 4.1. However, there will be a difference in the result obtained by executing the program P on sample when compared to the result obtained by executing the program P on non-reduced input data. This difference in result is tolerable for few applications of provenance. For example, in Debugging of a data flow process. Data samples are passed through every operator within the process during its execution and checked if they work as expected. The result of the entire non-reduced input data is not required in this scenario. The data sample is merely used to perform a sanity check on the process. Provenance is only collected for this sample of data to validate the flow of data across process execution.

We now apply stratified sampling to the example data in Table 4.1. Let column 5, “Country” be the attribute that defines the strata. Only two strata exist - the tuples which belong to “Alaska USA” form one strata and which belong to “Australia” form the other one. Certain application of this data may require that the rows of the sample should be uniformly distributed across these strata. For example, when calculating average temperature per country, from the example data of Table 4.1, it would be more sensible to have tuples belonging to every country, rather than having the possibility that random sampling might completely eliminate tuples from a particular country. We apply stratified sampling with strata defined by the attribute - Country and sampling ratio of 66% to the example data of Table 4.1. Figure 4.4 shows the stratified sample obtained. We see that the sampling is performed in such a way that 66% of tuples are selected from each strata. Hence we obtain 4 tuples each that contain values as “Alaska USA” and “Australia” for the

	Date	Latitude	Longitude	Temperature	Country
t_0	26-04-2016	65	-152	60	Alaska USA
t_1	26-04-2016	66	-153	54	Alaska USA
t_2	26-04-2016	65	-151	49	Alaska USA
t_3	26-04-2016	65	-152	54	Alaska USA
t_4	26-04-2016	-33	151	23	Australia
t_5	26-04-2016	-33	151	17	Australia
t_6	26-04-2016	-31	115	18	Australia
t_7	26-04-2016	-31	115	23	Australia

Table 4.4: Stratified sample of example data of Table 4.1 with sampling ratio of 66%

attribute Country, unlike random sampling, where we could have obtained any number of tuples with values as "Alaska USA" and "Australia" for the attribute Country.

Now we execute the same Spark program P from Section 4.1 on this sample. Provenance of the tuple t_a present in the result of the program P on data in Table 4.4 is now the set of tuples, t_0, t_1, t_2 and t_3. We see that the provenance now corresponds to 4 tuples instead of 6, as was the case with non reduced data in Table 4.1 where no sampling was performed. However, there will be difference in the result obtained by executing program P on the non-reduced input data and the stratified sample. This difference in result is tolerable for few use cases of provenance data. For example, in Logic testing of a data flow process, where only a few tuples are used for sanity check of the process instead of all the tuples in input. Provenance is collected only for these few tuples and analysis is performed on this reduced provenance.

4.3 Histogram analysis

Intuitively, a histogram is a plot that shows the frequency distribution of numerical data. It allows for inspection of the data for its underlying distribution, outliers, skewness, etc.

In the mathematical context, a histogram is a function m_i that counts the number of observations that fall into each of the disjoint categories. The graph of a histogram is merely one way to represent a histogram [Al-14]. The disjoint categories are called buckets or bins. The buckets are usually specified as consecutive, non-overlapping intervals and must be adjacent [AA03]. If we let n be the total number of observations and k be the total number of buckets, the histogram buckets m_i that constitute the histogram meet the following condition:

$$n = \sum_{i=1}^{k} m_i.$$

That is, if we are making a histogram of n data items of x variable, we want to construct a sequence of numbers m_1, m_2, ..., m_k such that the sum of all m_i is n itself (that is, we're counting numbers of x, not the actual values of x). There is no best way to find the size of the bucket. A simple way to calculate the number of buckets k given the bucket width h is:

$$k = \left\lceil \frac{\max x - \min x}{h} \right\rceil$$

To construct a histogram, we first split the entire range of values for a certain variable or attribute into a series of intervals. Then we count the number of values that fall into each interval. Figure 4.4 illustrates a histogram. The given data lies between the range 0 and 19. For a bucket size of 4, we divide the data into 5 buckets - [0-3], [4-7], [8-11], [12-15] and [16-19]. The number of occurrences that fall into each of these buckets represents their respective frequencies. We can see that the bucket [8-11] has the highest frequency, i.e., this bucket contains most data items.

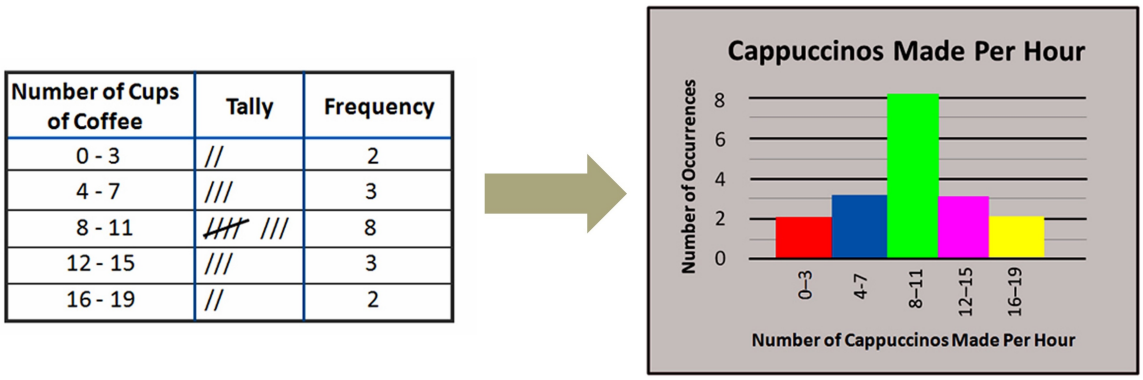


Figure 4.4: A Histogram [Pas12]

Histogram analysis for provenance data reduction

We use histograms as a provenance data reduction technique. We assume that the histogram bucket with largest frequency contains most information about the input data and is the most interesting bucket as it contains most frequently occurring data items. Once we obtain buckets by performing histogram analysis, we use them to collect provenance for only a subset of input data in the following ways:

1. **Reduction by eliminating buckets**: We consider only the bucket with highest frequency and collect provenance only for data items that belong to that bucket.
2. **Reduction by eliminating data items from each bucket**: From each of the buckets obtained, we select randomly few data items and collect provenance only for them. Hence we refrain ourselves from collecting provenance of all of input and use only reduced data obtained from histogram buckets.

Consider the example data from Table 4.1. We know the use cases or applications the provenance of this input data would be used in. That is why we have studied the many applications of provenance in Chapter 3. We know that the data is being used to measure the average temperature per country and perform analysis based on this. Histograms give us the frequency distribution of data and we will retain provenance for those tuples which reside in the bucket with most frequent temperature values as the reduced dataset. As we are measuring average, the difference between the result obtained from reduced dataset and the input data would be less. This is because, frequently occurring temperature values would affect the average more than the non frequently occurring ones.

We create histogram of temperature values from example data from Table 4.1. The width of the bucket is specified to be 5. The maximum and minimum values of temperature are 17 and 60 respectively. We calculate the number of buckets k by using the equation mentioned above and obtain 8 buckets - [17-22, [23-28, [29-34, [35-40, [41-46, [47-52, [53-58 and [59-64. Table 4.5 a) shows the values of temperature from example data of Table 4.1. Figure 4.5 b) shows the frequency count for every bucket.

51
39
60
18
20
54
49
54
23
17
18
23

a) Temperature values

Frequency Table	
Bucket	**Count**
17-22	4
23-28	2
29-34	0
35-40	1
41-46	0
47-52	2
53-58	2
59-64	1

b) Frequency table for temperature values

Figure 4.5: Range of values and Frequency table

Figure 4.6 illustrates the histogram produced for the frequency table in Figure 4.5 b). We see that the bucket [17-22 contains most of the temperature values. The tuples corresponding to this bucket are t_3, t_4, t_9 and t_{10} from Table 4.1. We only consider these tuples as reduced dataset for collection of provenance. We run the program P in Section 4.1 on this reduced dataset. The amount of provenance collected corresponds to only 4 tuples instead of 12 which is 33% of the size of input data. Use cases like Data summaries described in Section 3.3 can use histogram analysis to reduce provenance data. These use cases do not require the complete result from the reduced data to be accurate when compared to the input data. Data summaries involve computing histograms of reduced dataset on a certain transformation(s) for multiple data flow

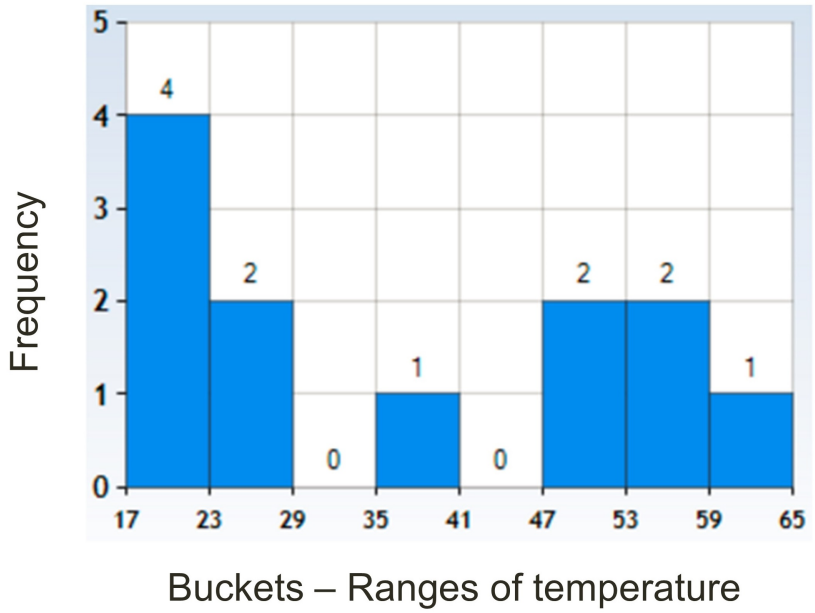


Figure 4.6: Histogram for temperature values for example data in Table 4.1

process executions. These histograms are compared against histograms computed on previous executions to find if there are any major data distribution changes. The data distribution changes might have stemmed from a failure in the transformation or faulty data, and the process has to be examined.

We collect provenance only for the tuples lying in the histogram bucket with highest frequency. This will reduce the amount of collected provenance as we are considering only a subset of data tuples and not all input tuples.

4.4 Clustering

Cluster analysis or clustering is the task of grouping a set of objects in such a way that objects in the same group (called a cluster) are more similar to each other than to those in other groups (clusters) [PMS14]. The greater the similarity within the group and the greater the difference between groups, the better is the clustering [JMF99].

Cluster analysis can be performed using various algorithms that create clusters in different ways based on their definition of what cluster means. Popular notions of clusters include groups with small distances among the cluster members, dense areas of the data space, intervals or particular statistical distributions [LW15].

Clustering can be roughly distinguished as:

1. **Hard clustering**: In this type of clustering, each data item belongs to a single cluster.
2. **Soft clustering**: In this type of clustering, each data item can belong to one or more clusters and has a certain degree with which it belongs to those clusters.

Clustering algorithms can be categorized based on their cluster model. In centroid-based clustering, clusters are represented by a central representative vector, which may not necessarily be a member of the data set [JMF99]. For example, the k-means algorithm represents each cluster by a single mean vector. Figure 4.7 illustrates clustering performed using k-means method. Given k=3, three clusters are performed and centroids of each of the clusters are marked with X.

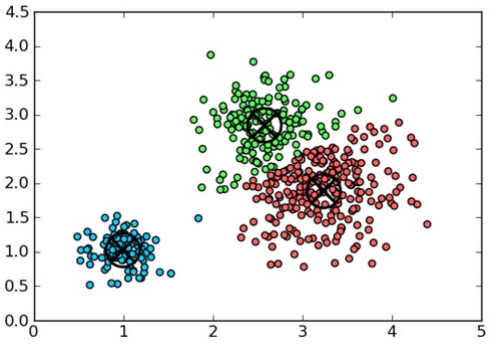


Figure 4.7: K-means clustering [Mac11]

K-means clustering algorithm aims to partition n data items into k clusters in which each data item belongs to the cluster with the nearest mean, which is the representative of the cluster. This results in a partitioning of the input data [Mac+67]. When the number of clusters is fixed to k, k-means clustering is an optimization problem to - find the k cluster centers and assign the data items to the nearest cluster center, such that the squared distances from the cluster are minimized. Formally, given a set of data items (x_1, x_2, ..., x_n), where each data item is a d-dimensional real vector, k-means clustering aims to partition the n data items into k ($\leq$ n) sets S = S_1, S_2, ..., S_k so as to minimize the within-cluster sum of squares (sum of distance functions of each data item in the cluster to the K center). In other words, its objective is to find:

$$\arg\min_{\mathbf{S}} \sum_{i=1}^{k} \sum_{\mathbf{x} \in S_i} \|\mathbf{x} - \boldsymbol{\mu}_i\|^2 \arg\min_{\mathbf{S}} \sum_{i=1}^{k} \sum_{\mathbf{x} \in S_i} \|\mathbf{x} - \boldsymbol{\mu}_i\|^2$$

where μ_i is the mean of data items in S_i [Mac+67].

	Date	Latitude	Longitude	Temperature	Country	Cluster ID
t_0	26-04-2016	66	-153	51	Alaska USA	1
t_1	26-04-2016	68	-151	39	Alaska USA	1
t_2	26-04-2016	65	-152	60	Alaska USA	1
t_3	26-04-2016	-16	145	18	Australia	2
t_4	26-04-2016	-16	140	20	Australia	2
t_5	26-04-2016	66	-153	54	Alaska USA	1
t_6	26-04-2016	65	-151	49	Alaska USA	1
t_7	26-04-2016	65	-152	54	Alaska USA	1
t_8	26-04-2016	-33	151	23	Australia	3
t_9	26-04-2016	-33	151	17	Australia	3
t_{10}	26-04-2016	-31	115	18	Australia	3
t_{11}	26-04-2016	-31	115	23	Australia	3

Table 4.5: K-means clustering on example data of Figure 4.1

K-means clustering for provenance data reduction

We use k-means clustering algorithm as a provenance data reduction technique. We first perform k-means clustering on input data and create clusters. Then there are two ways we reduce the data.

1. **Reduction by eliminating clusters**: We consider only a finite subset of clusters instead of all and collect provenance only for data items that belong to those few clusters.
2. **Reduction by eliminating data items**: From each of the clusters, we randomly select few data items and collect provenance only for them.

Consider the example data from Table 4.1. We form clusters in this data based on the latitude and longitude attributes of the tuples. Given number of clusters as k=3, Table 4.5 shows the three clusters formed and their corresponding cluster IDs.

The Spark program P from Section 4.1 computes average temperature for every country and collects provenance for the tuples present in the result of this query. If the application of provenance data is interested in only the tuples belonging to northern part of Australia, we do not have the information on it as the attribute “Country” only specifies the name of the country. The analyst of the application may notice that we have two clusters produced for the country “Australia”. Examining the value of the attribute “Latitude” in the tuples within Cluster 2, she discovers that the tuples correspond to northern Australia. We now consider only the tuples in cluster with cluster ID as 2. Table 4.6 shows the data reduced. We are now not collecting provenance for the other tuples belonging to other regions as we have reduced the data based on the application of provenance.

t_3	26-04-2016	-16	145	18	Australia
t_4	26-04-2016	-16	140	20	Australia

Table 4.6: Reduced data belonging to only cluster 2

	Date	Latitude	Longitude	Temperature	Country	Cluster ID
t_1	26-04-2016	68	-151	39	Alaska USA	1
t_2	26-04-2016	65	-152	60	Alaska USA	1
t_3	26-04-2016	-16	145	18	Australia	2
t_6	26-04-2016	65	-151	49	Alaska USA	1
t_7	26-04-2016	65	-152	54	Alaska USA	1
t_8	26-04-2016	-33	151	23	Australia	3
t_9	26-04-2016	-33	151	17	Australia	3

Table 4.7: Reduced data after randomly sampling 66% of tuples from every cluster

Another way we could reduce the data using clustering is by randomly selecting few tuples from every cluster and collecting provenance only for them. We have obtained the tuples belong to each cluster. Table 4.5 shows this. On these clusters, we apply random sampling with sampling ratio of 66% to produce reduced data. Table 4.7 shows the reduced data obtained after random sampling on clusters.

Provenance is now collected for tuples belonging to all the clusters. The result contains tuples that belong to any country, and not just northern Australia as was required by the application. The provenance of the tuple t_a in the result set corresponds to 4 tuples - t_1, t_2, t_6 and t_7. As we collect only a certain number of tuples from each cluster and not all of the input data, the size of provenance collected will be less than size of provenance collected for the entire input.

4.5 Equivalence classes

Equivalence classes try to divide input data into different groups, where the data items of a group are similar to each other in some aspect - a value, a property or meaning. That aspect is formalized by equivalence relation. In mathematics, an equivalence relation is a binary relation that is at the same time a reflexive relation, a symmetric relation and a transitive relation [SEA14]. A simple example of equivalence relation is “Has the same birthday as” on the set of all people and “has the same colour as” on set of all cars. Equivalence relations on sets create equivalence classes. These classes are disjoint. Equivalence classes on the equivalence relation “has the same colour as” on set of all cars are - cars with red color, cars with blue color, cars with green color and so on.

Equivalence partitioning is a technique to divide the input data into different equivalent classes. To obtain equivalent classes, we partition the input data into groups based on value taken by a certain attribute. In Figure 4.8, we produce four groups in data based on the value of the attribute "payer". The tuples that contain "Kodyaz" as the value are part of the group GR1, the ones that contain "SQL Trainer" belong to GR2 and so on. These groups GR1, GR2, GR3 and GR4 can be considered as equivalent classes.

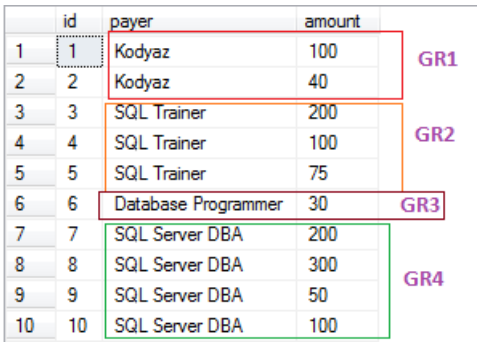


Figure 4.8: Groups in data based on the value of attribute "payer"

Equivalence Classes for provenance data reduction

We use equivalence classes as data value reduction technique. We first create equivalent classes in input data based on certain attribute value and then reduce the amount of provenance collected in two ways.

1. **Reduction by eliminating classes**: We consider only a finite subset of equivalent classes instead of all and collect provenance only for data items that belong to those few classes.
2. **Reduction by eliminating data items from every class**: From each of the equivalent classes created, we randomly select few data items and collect provenance only for them. We assume that the data items within each group are equivalent to each other and hence collecting provenance for a few data items from each group would provide the application or use case of provenance data with required information.

Consider the example data from Table 4.1. We divide the data into groups based on the attribute "Country" because we assume that temperatures across a country would not vary much and the values would be equivalent to each other. And also, we are interested in computing average temperature for each country, hence ignoring a few tuples from every country should not affect the average significantly. The groups with tuples belonging to a certain country form the equivalent classes. Table 4.8 shows the equivalence class corresponding to every tuple in the data.

We collect data only from Class 2 from data from Table 4.8. Table 4.9 shows the reduced data that corresponds to tuples with Country as "Australia". The provenance of the tuple t_b corresponds to 6 tuples - $t_3, t_4, t_8, t_9, t_{10}$ and t_{11}. Provenance now corresponds to these 6 tuples,

	Date	Latitude	Longitude	Temperature	Country	Class
t_0	26-04-2016	66	-153	51	Alaska USA	1
t_1	26-04-2016	68	-151	39	Alaska USA	1
t_2	26-04-2016	65	-152	60	Alaska USA	1
t_3	26-04-2016	-16	145	18	Australia	2
t_4	26-04-2016	-16	140	20	Australia	2
t_5	26-04-2016	66	-153	54	Alaska USA	1
t_6	26-04-2016	65	-151	49	Alaska USA	1
t_7	26-04-2016	65	-152	54	Alaska USA	1
t_8	26-04-2016	-33	151	23	Australia	2
t_9	26-04-2016	-33	151	17	Australia	2
t_{10}	26-04-2016	-31	115	18	Australia	2
t_{11}	26-04-2016	-31	118	23	Australia	2

Table 4.8: Groups in example data from Table 4.1 based on the value of attribute Country

	Date	Latitude	Longitude	Temperature	Country	Class
t_3	26-04-2016	-16	145	18	Australia	2
t_4	26-04-2016	-16	140	20	Australia	2
t_8	26-04-2016	-33	151	23	Australia	2
t_9	26-04-2016	-33	151	17	Australia	2
t_{10}	26-04-2016	-31	115	18	Australia	2
t_{11}	26-04-2016	-31	118	23	Australia	2

Table 4.9: Reduced data after retaining tuples from class 2

however we do not collect provenance for result tuple t_a, which was not the case for non-reduced input data.

We apply random sampling with sampling ratio of 66% on the classes from data from Table 4.8 to produce the reduced data. Table 4.10 shows the reduced data obtained after random sampling on the classes. The provenance of the tuple t_a corresponds to 4 tuples - t_0, t_1, t_5 and t_6. Provenance now corresponds to these 4 tuples rather than 6, as was the case with non-reduced input data.

4.6 Deduplication

Data deduplication is the process of eliminating duplicate data from the input. It is a common data compression technique and is used to save storage space. Deduplication identifies copies of data items that are not unique and ultimately stores only one copy. This ultimately creates a

	Date	Latitude	Longitude	Temperature	Country	Class
t_0	26-04-2016	66	-153	51	Alaska USA	1
t_1	26-04-2016	68	-151	39	Alaska USA	1
t_3	26-04-2016	-16	145	18	Australia	2
t_4	26-04-2016	-16	140	20	Australia	2
t_5	26-04-2016	66	-153	54	Alaska USA	1
t_6	26-04-2016	65	-151	49	Alaska USA	1
t_{10}	26-04-2016	-31	115	18	Australia	2
t_{11}	26-04-2016	-31	118	23	Australia	2

Table 4.10: Reduced data after randomly sampling 66% of tuples from every class in data from Table 4.8

reduced dataset that contains only distinct data items [MB12]. Larger the number of occurrences of non-unique data items, smaller is the size of reduced dataset.

We can use data deduplication as a provenance data reduction technique. We can collect provenance only for the unique data items present in the input data instead of all of the input. Thus, we reduce the total amount of provenance collected for that input. However, if there are no duplicate data items present, there will be no reduction and hence we collect the same amount of provenance for unique data set as for the input data.

4.7 Outlier detection

"An outlier is an observation which deviates so much from the other observations as to arouse suspicions that it was generated by a different mechanism" [Haw80]. Normal data objects behave in a certain fixed manner, abnormal objects deviate from this behaviour. Generally the data items corresponding to abnormal behaviour will relate to some kind of problem in the system. Outliers are also referred to as noise, exceptions and anomalies.

For example, when measuring temperature of a room whose temperature normally lies between 10 degrees Celsius and 40 degree Celsius, a temperature reading of 75 degrees Celsius would mean that it is an outlier. This outlier may have been caused due to numerous scenarios like fire, smoke, faulty device etc., all of which require attention as they are not part of normal operation. Another example, when collecting banking transactions data of a customer, who usually credits and debits in 100s of euros, a debit of 10000 euros would have to be checked. This transaction is out of normal behaviour of the customer and hence would need verification from the bank to validate the identity and avoid any fraud.

Figure 4.9 shows how an outlier that is present in the data set can be visualized when plotted as a graph. Outlier does not adhere to the pattern followed by other data items and is away from rest of them.

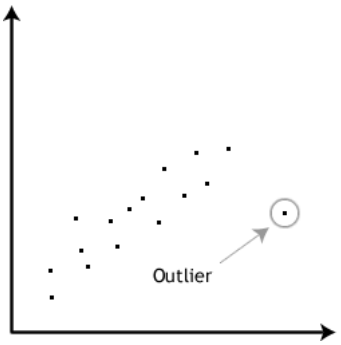


Figure 4.9: Outlier in a dataset [Haw80]

Outlier detection can serve as a provenance data reduction technique where we find outliers and collect provenance only for the outliers. We assume that these outliers contain most interesting information required by the application of provenance data. Outliers typically represent some type of problem like erroneous data, fraudulent transaction or invalid data. When using provenance for debugging or monitoring of processes, it might be necessary to find this anomalous data and collect provenance for the same. In case of example of temperature measurements, the provenance pertaining to the anomalous reading would lead us to the device and the location and we can quickly check what has caused the reading to be abnormal. In case of the example of recording bank transactions, the provenance collected for the abnormal transaction would provide information on where the withdrawal took place, which debit card was used etc. This information helps the bank to identify if the transaction was a fraud or that the customer himself withdrew the large amount. Because we are collecting provenance only for outliers in this technique and not for all the data items in the input, the size of provenance is significantly reduced, which is our main intention.

4.8 Stream summary

Stream-Summary algorithms belongs to the genre of algorithms and data structures that estimate the frequency of data items. Stream-Summary detects the most frequent data items from input and estimates their frequencies with explicitly tracked estimation error.

Stream-Summary tracks a fixed number of data items that presumably are most frequent ones. This fixed number corresponds to number of “slots”. For example, if number of slots is 10, stream summary gives 10 most frequently occurring data items. If one of these data items occurs in the stream of data, the counter corresponding to that data item is increased. If a new, non-tracked data item appears, it replaces the least frequent tracked data item and the eliminated data item become non-tracked [MAEA05].

Stream-Summary assigns data items to a group that has same frequency, i.e. to the number of occurrences. Also, each tracked data item has the "error" attribute that gives the estimation error. Ultimately we get "n" most frequently occurring data items, where "n" is the number of slots. We can collect provenance only for these "n" frequently occurring data items instead of all of input data and hence reduce the overall amount of provenance collected.

4.9 Count-Min Sketch

Another frequency estimation algorithm we can use in data value reduction is Count-Min Sketch. Count-Min Sketch not only estimates frequency of data items but also estimates frequency-related properties of the data set, e.g. find top-K frequent elements, perform range queries (where the goal is to find the sum of frequencies of elements within a range), estimate percentiles. Count-Min Sketch works well for frequent values and estimates their frequency-related properties with good accuracy, however estimations for relatively rare values can be imprecise [CM05].

Figure 4.10 illustrates the basic idea of Count-Min Sketch. Count-Min sketch produces a two-dimensional array (d x w) of integer counters. When a value arrives, it is mapped to one position at each of d rows using d different and preferably independent hash functions. Counters on each position are incremented. Count–Min sketch is a probabilistic data structure can be treated as a frequency table of data items in a stream of data. It uses hash functions to map data items to their frequencies [Ily12].

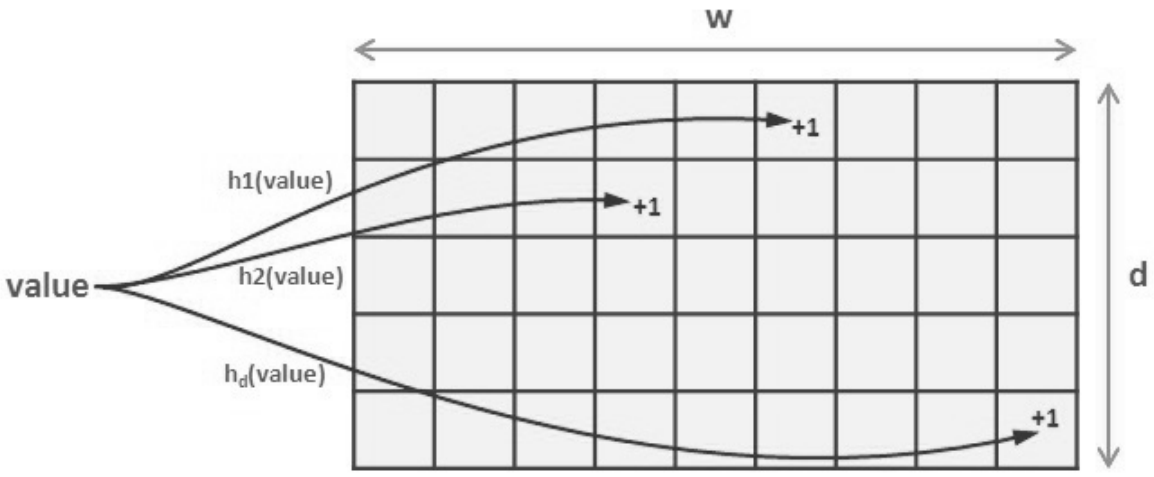


Figure 4.10: Count-Min Sketch [Ily12]

Count-Min Sketch provides frequencies for data items present in input data. We can collect provenance for only certain finite set of "n" most frequent data items assuming that they contain most important information regarding the entire input data. Hence, we reduce the amount of

provenance collected by not tracking provenance for all data items, and instead only the most frequent ones.

4.10 Locality Sensitive Hashing and Min Hash

A locality sensitive hash (LSH)[LRU14] is a hash function that takes some data item and hashes it into a number, with the added condition that two items that are close by some metric of interest in the original space have hashes that are close to the same value. This is very different from a commonly understood hash function, where the value of hash(200) does not have any relation to value of hash(201). In LSH, value of LSH(200) and LSH(201) will be closer to each other than, foe example the value of LSH(500). It hashes input items so that similar items map to the same "group" with high probability. LSH can be treated as an alternative to clustering, as clustering is pretty expensive computationally. A good LSH will scale linearly with the number of data items. LSH reduces the number of dimensions of data.

Min Hash It converts large data sets into short signatures while preserving similarity [LRU14]. We can combine Min Hash and LSH. The short signatures provided by Min Hash are used by LSH to group similar items into same buckets. These buckets will contain closely related data items.

We can use LSH and Min-Hash together as a data value reduction technique to form groups containing similar data items out of input data. We can then select randomly few data items from every group and collect provenance for them. Assuming that all data items within the same group are same, tracking provenance for only a few of them would be sufficient for answering provenance queries executed on top of them.

4.11 Discussion

All the provenance data reduction techniques mentioned above can be applied to the use cases and applications of provenance which do not require information present in every data tuple present in the input. They can be applied data-agnostic use cases and applications of provenance. The following is the list of such use cases and applications. These use cases and applications are described in Chapter 3:

- Performance overhead profiling
- Latency profiling
- Memory usage profiling
- Data quality of overall data based on sample
- Debugging

- Context information
- Data summaries
- Data Samples
- Trial runs based on sample
- Data exploration

5 Implementation and Evaluation

This chapter is dedicated to evaluation of simple random sampling, histogram analysis, k-means clustering and equivalence classes as provenance data reduction techniques. These techniques collect partial provenance for only a subset of input data and not all of the input. As of now there is no benchmark available to compare different provenance solutions. Hence, we use two datasets and define multiple scenarios on them to test our provenance data reduction techniques. We also provide the provenance queries and the applications of provenance related to these scenarios. We then provide details on our test setup and describe how we collect provenance using a library called Titian. We evaluate the provenance data reduction techniques to examine their effectiveness, in comparison to one another. For that purpose we introduce and formally define quantitative metrics. Then we apply the data reduction techniques from the previous chapter to collect partial provenance, and evaluate each reduction technique with the help of these metrics. Towards the end of this chapter, we perform a qualitative analysis where we evaluate to which degree the partial provenance suffices to answer the provenance queries from the scenarios.

5.1 Datasets and scenarios

In this section, we describe the two datasets - Safecast radiation dataset and U.S Domestic flights dataset and the scenarios executed on them as part of our evaluation. Every scenario description contains an introduction to the scenario, the Apache Spark [The16] program that we use to realize the scenario, the application of provenance from Chapter 3 the scenario can be used in and the provenance question that is executed on the provenance of the result set of this Spark program. The scenarios collectively address four main categories of the applications of provenance from Chapter 3 - debugging, data exploration, monitoring and data quality. These applications or use cases of provenance do not always require fine grained provenance regarding every tuple of input data; provenance for a subset of input data is sufficient. Thus these applications provide potential for provenance data reduction. We collect provenance for the result of the Spark programs using Titian [Int+15], a library that allows fine grained provenance tracking in Spark programs. We also use the same library in collecting reduced provenance data. In Section 5.3, we describe the provenance collection within Spark programs using Titian in detail.

5.1.1 Safecast radiation dataset and scenarios executed on it

Safecast [Saf16] is a website that collects data about environmental radiations and allows users to access it for free. We used their data that provides information on radiation based on the

Captured Time (0)	Latitude (1)	Longitude (2)	Value (3)	Unit (4)	Location Name (5)	Device ID (6)
2010-02-03 17:00:00	37.50	139.94	72	cpm	0	0
MD5Sum (7)	Height (8)	Surface (9)	Radiation (10)	Uploaded Time (11)	Loader ID (12)	
6449bbf7	0	0	0	2010-02-03 17:00:00	633	

Table 5.1: Attributes of Safecast radiation dataset

latitude and longitude of the location in our evaluation. The data can be downloaded as a comma-delimited flat file. Table 5.1 shows a sample row of data and describes the columns present in it. In the next section, we describe the scenarios for the Safecast dataset.

Scenario 1

This scenario requires to present 10 highest radiation levels per year from 2010 to 2016. The results are displayed in descending order. To realise the scenario, we use an Apache Spark program:

```
1 val tuples = Data.map(_.split(","))
2           .filter(p => p(0).length == 19)
3           .filter(p => (p(0).substring(0,4)).toInt <= 2016)
4           .filter(p => (p(0).substring(0,4)).toInt >= 2010)
5
6 val kvpair = tuples.map(p => (p(0).substring(0,4),p(3).toDouble))
7 val group = kvpair.keyBy(r => (r._1)).groupByKey
8 val result = group.mapValues(iter => iter.toList.sortBy(- _._2).take(10))
9 result.collect()
```

Statement 1 in the program first separates every tuple of the data file into an array of multiple columns by splitting at "," and then filters out invalid data from column 0 whose length is not equal to 19. From this data, we extract only the year part from the date present in column 0 and select only those tuples that are from years 2010 to 2016. This array of multiple columns is then assigned to the reference "tuples". Statement 6 extracts only the year and column 3 that contains the radiation reading and creates a key value pair of it and assigns it to reference "kvpair". Statement 7 groups the tuples by their years and assigns the data to reference "group". Statement 8 sorts the data for each year and then extracts only the 10 highest radiation values and assigns it to reference "result". The collect action in Statement 9 triggers the evaluation of "result" reference and all the transformations leading up to it. This program contains two map operators and one reduce operator, and one filter operator on the input data that considers

Year	Radiation values
2016	64210.0, 64210.0, 64210.0, 64210.0, 64210.0
2015	39636.0, 17155.0, 16638.0, 12504.0, 9890.0
2014	19473.0, 19345.0, 18928.0, 18007.0, 17995.0
2013	57719.0, 37534.0, 19557.0, 13335.0, 12572.0
2012	24210.0, 20233.0, 20220.0, 10640.0, 7281.0
2011	79458.0, 77111.0, 77241.0, 76123.0, 76002.0
2010	9128.0, 7659.0, 7201.0, 6543.0, 6439.0

Table 5.2: Result of Spark program of scenario 1

only the tuples belonging to years between 2010 and 2016. Table 5.2 shows the result obtained by executing the Spark program. Because of shortage of space, we have shown only 5 highest radiation values of each year, instead of 10.

The result shows that there is a sudden increase in radiation levels in 2011 and that the 10 highest values from 2011 are very high when compared to that of other years. Chapter 3 describes many use cases and applications of provenance. An application of this scenario lies in data exploration where we want to know the change in radiation levels across the years and find interesting characteristics within the underlying data. Examining the result, we pose a question - why is there a sudden icrease of radiation level in the year 2011. In order to answer this question, we make use of the collected provenance data. We inspect the provenance associated only with the tuples belonging to year 2011 and see that the radiation levels are high specifically for the months after March. We then look at the latitude and longitude of the tuples from March 2011 that have very high radiation values, and find that they belong to the region Fukushima in Japan. We find the answer to this query that these tuples that show the highest 10 radiation readings in 2011 correspond to the Fukushima nuclear disaster that occurred due to the tsunami and earthquake that hit Japan at that time.

We examine the results of the Spark program and also notice that the 10 highest values are all equal to one another for the year 2016. In addition these values are close to those of 2011, even when there was no major nuclear disaster in 2016. We suspect this may be due to some erroneous data. In addition to data exploration, this observation in results helps us find invalid or erroneous data. Thus, this scenario is useful in data quality, too. Data quality is one of the many applications of provenance which are described in Chapter 3. Based on the observation, we like to find out - why are all the 10 highest values from 2016 equal to one another? We first consider the full provenance for the result tuples and selectively examine the provenance for tuples belonging to the year 2016. The provenance showed moderately high values for most of the tuples other than the few tuples with very high radiation readings. We then filter on this data to only examine the tuples with very high radiation readings. We find that all the tuples that have the high values have the same location and the readings remain the same for few hours. The location corresponding to these readings does not have any major nuclear event. We conclude that such readings probably are from a faulty device. Thus, this question is useful in Data quality,

where in we examine the results and find anomalous tuples that correspond to invalid data. We then filter these tuples and re-run the program on the data and obtain appropriate results for all the years including 2016.

Scenario 2

This scenario aims at finding the average radiation level in the U.S per year and per month. Since we do not have an attribute in data that specifies the country directly, we use the latitude and longitude attributes to decide if the location is within the U.S or not. The following is a snippet of the Spark program executed for this purpose:

```
1  val tuples = Data.map(_.split(","))
2             .filter(p => p(0).length == 19)
3             .filter(p => p(2).toDouble >= 30.0)
4             .filter(p => p(2).toDouble < 50.0)
5             .filter(p => p(1).toDouble >= 66.0)
6             .filter(p => p(1).toDouble < 124.0)
7  val kvpair = tuples.map(p =>
       (p(0).substring(0,4).concat(p(0).substring(5,7)),p(3).toDouble))
8  val avg = kvpair.mapValues(x => (x, 1))
9            .reduceByKey((x, y) => (x._1 + y._1, x._2 + y._2))
10           .mapValues(x => (x._1/x._2))
11 avg.collect()
12 val kvpairYear = avg.map(p => (p._1.substring(0,4),p._2.toDouble))
13 val avgYear = kvpairYear.mapValues(x => (x, 1))
14              .reduceByKey((x, y) => (x._1 + y._1, x._2 + y_2))
15              .mapValues(x => (x._1/x._2))
16 avgYear.collect()
```

Statement 1 in the program splits every tuple from data into an array of multiple columns and then filters out tuples that contain invalid dates from column 0 whose length is not equal to 19. From this data, we extract only the tuples that correspond to the U.S based on the filter ranges provided on latitude (column 1) and longitude (column 2) columns and assign it to reference "tuples". Statement 7 extracts only the year and month part from the date(column 0) column and creates a key value pair that contains year and month as key and radiation reading as the value. This key value pair is assigned to reference "kvpair". Statement 9 calculates the average radiation for each year and month and assigns it to the reference "avg". Statement 12 contains the collect action that executes all the statements before it and presents the average radiation per year and month. Statement 13 uses the reference "avg" and creates a new key value pair that contains only the year as the key and the radiation reading as the value and assigns it to the reference "kvpairYear". Statement 14 computes the average radiation value based on the key values, that is the year and assigns it to the reference "avgYear". Statement 15 executes the references required to compute the final result and presents the average radiation levels in the U.S per year. This spark program consists of 4 map operators and 2 reduce operators along with

Year	Radiation values
2006	892.0
2016	32921.0
2015	11470.5
2006	116.0
1980	6639.0

Table 5.3: Result of Spark program of scenario 2

filter operator that only accepts tuples belonging to the U.S. Table 5.3 shows the few tuples from the result of the Spark program. The results contain data from years between 1980 and 2016.

In this scenario, we are interested in a specific event from 2006 where there was a nuclear accident in Tennessee USA. We expect the radiation levels in 2006 to be higher than usual in U.S due to this event. However we notice from the result of the Spark program that they are not. The observation poses a question - Why is the average radiation level of 2006 not high when compared to that of other years even when we know there was a nuclear spill in 2006? This question is useful in data exploration, where we find interesting insights about the underlying data based on its important properties. Also, we will see further that it is useful in data quality, too. Chapter 3 describes data exploration and data quality and many applications where provenance is useful. We use provenance collected as part of execution of Spark program to answer the question posed. We examine the provenance for only the tuples in 2006 and find that there are only a limited amount of tuples for 2006. These tuples are not sufficient to present any valuable insight about the accident because they do not correspond to the location Tennessee, where the accident occurred. Hence the data does not have complete information necessary to answer the question. This violates the "completeness" condition of data quality as defined in Section 3.4. Thus we use provenance in data quality. This question also has an aspect of data exploration if it contains sufficient data to provide information on the nuclear disaster in U.S that took place in 2006. In that case, it would provide insights based on interesting events present in the underlying data.

Scenario 3

This scenario is about the average radiation value per unit type. The unit type is provided in column 4 in the data. We execute an Apache Spark program to find the result:

```
1 val tuples = Data.map(_.split(",")).filter(p => p(0).length == 19)
2 val kvpair = tuples.map(p => (p(4),p(3).toDouble))
3 val avgByUnitType=kvpair.mapValues(x => (x, 1))
4              .reduceByKey((x, y) => (x._1 + y._1, x._2 + y._2))
5              .mapValues(x => (x._1/x._2))
6 avgByUnitType.collect()
```

Unit type	Average radiation value
DeviceType2	130.0
microsievert	124.2
cpm	659.0
status	22.5
DeviceType1	129
usv	0.3

Table 5.4: Result of Spark program of scenario 3

Statement 1 in the program creates an array of multiple columns by splitting every tuple in data at "," and then removes invalid data from column 0 whose length is not equal to 19. It then assigns this data to reference "tuples". Statement 2 creates a key-value pair with unit type as the key and the radiation reading from every tuple and assigns it to reference "kvpair". Statement 3 computes the average radiation reading based on the key, i.e., unit type and assigns the result to reference "avgByUnitType". Statement 6 contains the collect action that executes the previous references and presents the final result. The Spark program has two map operators and one reduce operator. However we have no filter operator other than the one that eliminates invalid records which is identical to the one in the other Spark programs. This program considers all tuples from valid input data and all of this data is involved in producing the result of this program. Table 5.4 shows 6 tuples from the result of this Spark program that presents the average radiation level per unit type. Unit type refers to the unit of measurement used to record radiation. The result provides us with information on what units occur within the data. We inspect the result tuples and find that 10 unit types occur.

This scenario is useful in monitoring. Monitoring and many other applications of provenance are described in Chapter 3. We execute a program to count tuples that contain each of these types. The count program executed on unit type as "cpm" has higher execution time when compared to the execution times of the same program executed on other unit types. We raise the question - why does the program on "cpm" takes the longest? This question is helpful in monitoring. In latency profiling, we monitor the time taken by different datasets or data tuples to perform certain task. This time remains the same as long as similar execution environment and data are used. If the time differs largely from what it was in previous runs, there is a possibility that something is wrong either in the system or within the data. In both cases, we need to investigate what caused this.

5.1.2 U.S domestic flights dataset

U.S Department of Transporation's Bureau of Transportation Statistics tracks real time performance of domestic flights operated by different airlines within the U.S. and publishes the raw data as comma delimited files for public use. We use their data on flight delays as part of our evaluation to execute multiple scenarios and provenance queries to assess the performance of

Year (0)	Month (1)	Day of month (2)	Day of week (3)	Departure Time (4)	CRS Departure Time (5)	Arrival time (6)
2000	1	28	5	1647	1647	1906
CRS Arrival time (7)	Unique carrier code (8)	Flight Number (9)	Tail Number (10)	Actual Elapsed Time*(11)	CRS Elapsed Time*(12)	Air Time* (13)
1859	HP	154	N808AW	259	252	233
Arrival Delay*(14)	Departure Delay*(15)	Origin (16)	Destination (17)	Distance (18)	Taxi in* (19)	Taxi Out* (20)
7	0	ATL	PHX	1587	15	11
Cancelled (21)	Cancellation code (22)	Diverted (23)	Carrier Delay*(24)	Weather Delay*(25)	Security Delay*(26)	Late aircraft delay* (27)
0	NA	0	NA	NA	NA	NA

* In minutes.

Table 5.5: Attributes of U.S Domestic flights dataset

our data reduction techniques. The data available belongs to the years between 2000 and 2008. Table 5.5 shows a sample row from this data and describes the attributes present.

Scenario 4

This scenario is about the average arrival delay in minutes for every year for every airline carrier. Statement 1 in the Spark program below, first separates every tuple in the data file into an array of multiple columns by splitting at "," and then filters out invalid data from column 14 (arrival delay). It then assigns this data to reference "cols". Statement 2 creates a key-value pair with column 0 - year as the key and arrival delay from every tuple and assigns it to reference "kvpair". Statement 3 computes the average arrival delay based on the key, i.e., year and assigns the result to reference "avgDelayPerYear". Statement 6 contains the collect action that executes the previous statements and presents the final result. This Spark program contains two map operations and a single reduce operation. Table 5.6 shows 5 result tuples obtained by executing the Spark program from above. Each tuple shows the average delay associated with each year for every airline carrier.

```
1 val cols = Data.map(s => s.split(",")).filter(p=> !p(14).contains("NA"))
2 val kvpair = cols.map(p => (p(0).concat(p(8)),p(14).toDouble))
3 val avgDelayPerYear = kvpair.mapValues(x => (x, 1))
4                 .reduceByKey ((x, y) => (x._1 + y._1, x._2 + y._2))
5                 .mapValues(x => (x._1/x._2))
6 avgDelayPerYear.collect()
```

Year	Airline Code	Average delay
2005	WN	5.36
2006	AA	56.75
2008	EV	14.69
2006	UA	78.70
2005	CO	8.75

Table 5.6: Result of Spark program of scenario 4

This scenario is useful in debugging. We describe debugging as one of the applications of provenance in detail in Chapter 3. In the context of this scenario, we are interested in finding the tuples that cause most delays. To this end, we execute multiple provenance questions - Why is the average delay in 2006 the highest? What airlines causes the most delays? Is there an airport that causes a bottleneck in air travel? These questions collectively help us find the tuples in data that correspond to maximum delay, and not just the average delay which is the result obtained from the Spark program. We examine the provenance for only the input tuples belonging to year 2006 and sort on the airlines column (column 8) to find the airlines that has flights with most delays. We find that "AA" and "UA", the unique carrier code for American airlines and United airlines, have highest values for delays. We now filter on airline column and consider only the tuples that contain "AA" and "UA" and execute the following provenance question on this data - Why do "AA" and "UA" have high average delay values? We now sort the data on the column that shows the origin of the flight, i.e., column 16. We see that most tuples belong to the airport with code "ORD", that corresponds to the O'Hare airport, Chicago. We see that this is the airport that has most delays because it is a major hub for American airlines as well as United airlines. These questions together are useful in debugging as we ultimately find the airport that is responsible for causing most delays by backward tracing along the provenance collected for the result of the Spark program.

Scenario 5

In this scenario, we find the average departure delay in minutes based on the day of the week. Statement 1 in the Spark program below, creates an array of columns by splitting every tuple in data at "," . It then eliminates invalid data from column 15 (Departure delay) and assigns this filtered valid data to reference "cols". Statement 2 creates a key-value pair with column 0 - year concatenated with the column 3 - day of the week as the key and departure delay as value from every tuple and assigns it to reference "kvpair". Statement 3 computes the average departure delay based on the key, i.e., year and day of the week and assigns the result to reference "avgDelayPerYearPerDay". Statement 6 extracts day of the week from every tuple within the reference "avgDelayPerYearPerDay" and creates a new key value pair with day of the week as key and departure delay as value. Statement 8 calculates the average departure delay with respect to the key value - day of the week and assigns the result to reference "avgGroupYearDelay". Statement 11 contains the collect action that executes the previous references and presents the average departure for a day of the week. This Spark program contains three map operations and

two reduce operations. Table 5.7 shows the result of the Spark program. The days of the week are numbered from 1 - Monday, 2 - Tuesday and so on.

```
1  val cols = Data.map(s => s.split(",")).filter(p=> !p(15).contains("NA"))
2  val kvpair = cols.map(p => (p(0).concat(p(3)),p(15).toDouble))
3  val avgDelayPerYearPerDay=kvpair.mapValues(x => (x, 1))
4                   .reduceByKey((x, y) => (x._1 + y._1, x._2 + y._2))
5                   .mapValues(x => (x._1/x._2))
6  val groupDayDelay= avgDelayPerYearPerDay.
7                 map(p =>(p._1.substring(4,p._1.length()),p._2))
8  val avgDelayPerDay = groupDayDelay.mapValues(x => (x, 1))
9                   .reduceByKey((x, y) => (x._1 + y._1, x._2 + y._2))
10                  .mapValues(x => (x._1/x._2))
11 avgDelayPerDay.collect()
```

This scenario is helpful in monitoring as well as data exploration as described further in the text. Chapter 3 introduces and describes monitoring and data exploration as one of the many applications of provenance. We examine the result and notice that the delay is higher on Thursdays and Fridays. To find out why, we pose a question - Why is the delay higher on Thursdays and Fridays? We answer this question using provenance collected during the execution of the Spark program. We notice from the result that the delay on Thursdays and Fridays is always higher than the delay on the other days. We consider this to be the normal behavior. If the delay is less on Thursday and Friday than usual, we investigate further to know if there exists a particular event that may have caused this. In the case of data exploration, we try to find the relation between delay and day of the week and based on it, decide what days are to travel on such that we encounter minimum delays.

We also notice from the result of the Spark program that the delay is very low for the years 2001 and 2002. To find out the reason, we pose a question - why is the delay not high for the years 2001 and 2002 when compared to other years? We use provenance associated with the result of the Spark program to answer this question. Provenance is collected for the result that presents average delay per day of the week, per year. We perform a filter operation on this provenance data and consider only the tuples that belong to the years 2001 and 2002. On these tuples, we backward trace the provenance and inspect the corresponding input tuples. We see that most of the tuples have the canceled column set to 1. We check for the dates and see that such flights are post September when the 9/11 attacks took place. Also, the number of flights in 2001 and 2002 is less when compared to other years. This provenance question thus helps in data exploration where we analyze the underlying data and find the reason behind certain behavior, in our case, less delay in 2001 and 2002 when compared to other years.

Year	Day of the week	Average delay
2000	Monday	6.98
2000	Tuesday	4.15
2000	Wednesday	6.5
2000	Thursday	10.19
2000	Friday	16.91
2000	Saturday	3.4
2000	Sunday	6.8
2001	Thursday	3.5
2001	Friday	4.91
2002	Thursday	4.5
2001	Friday	3.91

Table 5.7: Result of Spark program of scenario 5

Scenario 6

In this scenario, we find the average arrival delay for every airport from the year 2002 until 2006. Statement 1 in the Spark program below, splits every tuple in the data file into multiple columns and then removes invalid data from column 14 (arrival delay). It then eliminates tuples that belong to years other than the years between 2002 and 2006 and finally assigns this data to reference "cols". Statement 5 creates a key-value pair with column 16 - airport as the key and arrival delay from every tuple and assigns it to reference "kvpair". Statement 6 computes the average arrival delay based on the key, i.e., airport and assigns the result to reference "avgDelayPerAirport". Statement 9 contains the collect action that executes the previous references and presents the final result. This Spark program contains two map operations and a single reduce operation. It also contains a filter operation that considers only tuples from years between 2002 and 2006. We present seven tuples from the result of the Spark program in Table 5.8. These tuples presents the average arrival delay for the given airports between the years 2002 and 2006.

```
1 val cols = Data.map(s => s.split(","))
2       .filter(p=>!p(14).contains("NA"))
3       .filter(p=>p(0).toInt >= 2002)
4       .filter(p=>p(0).toInt <=2006)
5 val kvpair = cols.map(p => (p(16),p(14).toDouble))
6 val avgDelayPerAirport = kvpair.mapValues(x => (x, 1))
7             .reduceByKey((x, y) => (x._1 + y._1, x._2 + y._2))
8             .mapValues(x => (x._1/x._2))
9 avgDelayPerAirport.collect()
```

Airport	Average delay
SBP	-4.71
OTZ	11.96
TLH	7.45
HNL	0.85
FAT	-5.98
DBQ	-4.63

Table 5.8: Result of Spark program of scenario 6

This scenario has its application in debugging and data cleaning. We describe debugging and data cleaning as applications of provenance in Chapter 3. The result of the Spark program shows that there are few airports that have average delay in negative. To understand why such is the case, we we raise the question - Why is average arrival delay for some airports in negative? We answer it with the help of provenance collected as part of the Spark program execution. We first consider only the tuples in the result that have negative value for average delay. We extract the tuples in the input and notice that the delay values have been in negative for large number of tuples. In this scenario, we are not interested in early arrival, which can be the reason why delay is in negative. We only want to know the airplanes that actually have a delay. Hence we can debug the scenario, where we map the negative delay values to 0 as they bias the average delay. Another option would be to clean the input data altogether and change the negative values for delay to 0. This question is thus helpful in data cleaning where we find the data that is invalid and remove it. It is also helpful in debugging as the invalid data causes incorrect results to be presented. Removing this invalid data and re-running the Spark program gives valid positive result for average delay.

5.2 Test setup

All the scenarios are executed on a system with Intel Xeon i7-4600M 2.90 Ghz dual-core CPU, 12GB RAM, running CentOS 6.4 64 bit. We use Spark 1.2.1 and scala 2.10.1 to run our scenarios. To collect provenance in Spark programs, we use Titian library which is built in Spark. We sample the datasets so as to create smaller test datasets of sizes 100MB, 500MB, 1GB and 1.5GB, the largest test dataset containing up to 14 million tuples.

5.3 Collection of provenance

We use a library called Titian to collect provenance for the results of the spark programs above. Titian integrates with Spark programming model and provides a RDD called LineageRDD that facilitates provenance collection within Spark programs. The method getLineage provides a LineageRDD reference which acts like a starting point for tracing. Basic native RDD transformations like map, reduce, filter etc., can be used on this LineageRDD and provenance can be collected for

the data corresponding to these operators during the execution. However, provenance collection is not supported for operators like union, cogroup. For tracing of the results, Titian provides methods that can be executed on top of LineageRDD reference. goBack() and goBackAll() are useful in backward tracing, whereas goNext() and goNextAll() are used for forward tracing [Int+15].

The Spark code snippet below briefly illustrates collection of provenance for a result using Titian. Statement 1 shows the result of scenario 3 executed on U.S Domestic flights dataset. We collect provenance for the tuples in the result. Statement 2 calls the getLineage method on this reference "avgDelayPerAirport" which returns a LineageRDD with reference as "linRdd". We can save the provenance associated with the "avgDelayPerAirport" reference by saving it as object file or text file. Statement 3 saves the provenance as object file. We can use the backward tracing function goBack() to go back one step in execution and collect provenance and view it. Statement 5 contains the show() method that shows provenance associated with the reference "avgDelayPerAirport".

```
1 avgDelayPerAirport.collect()
2 var linRdd = avgDelayPerAirport.getLineage()
3 linRdd.saveAsObjectFile("/home/user1/provenance/")
4 linRdd=linRdd.goBack()
5 linRdd.show()
```

We first collect provenance for result of the spark program executed on non reduced input data and save it as an object file on disk. We then collect reduced provenance data using different data reduction techniques. Finally we measure quantitative metrics, described in next section, on these provenance data and compare them. These metrics describe the amount of provenance reduction against the information loss due to reduction. To this end, in Section 5.4.5, we perform detailed comparison between reduced provenance data obtained from different data reduction techniques.

5.4 Quantitative Analysis

As stated earlier, we use Titian library for provenance collection within Spark programs. We first execute the Spark program and collect provenance for its result tuple during the execution. In this case, we collect full provenance. That is, the provenance collected contains information on all the tuples in the result of the program, and also on all the tuples from the input that contributed to the tuples in the result. However, when we apply the provenance data reduction techniques - random sampling, histograms, clustering and equivalence classes, we collect partial provenance in order to reduce space overhead introduced by collection of full provenance. We use the same Titian library to collect partial provenance as well. However, Titian does not provide means to collect partial provenance in its current version. It does not allow us to apply the reduction techniques directly on the provenance data.

	Date	Latitude	Longitude	Temperature	Country
t_0	26-04-2016	66	-153	51	Alaska USA
t_1	26-04-2016	68	-151	39	Alaska USA
t_2	26-04-2016	65	-152	60	Alaska USA
t_3	26-04-2016	-16	145	18	Australia
t_4	26-04-2016	-16	140	20	Australia
t_5	26-04-2016	66	-153	54	Alaska USA
t_6	26-04-2016	65	-151	49	Alaska USA
t_7	26-04-2016	65	-152	54	Alaska USA
t_8	26-04-2016	-33	151	23	Australia
t_9	26-04-2016	-33	158	17	Australia
t_{10}	26-04-2016	-31	115	18	Australia
t_{11}	26-04-2016	-31	118	23	Australia

Table 5.9: Example data: Temperatures at different locations

To address this shortcoming, we apply the provenance data reduction techniques on the input data instead. During the execution of the spark program, we apply the reduction techniques to the input data and reduce the input data. This enables Titian to collect partial provenance as it now deals with reduced input data. We apply all four data reduction techniques to the input data and collect four corresponding sets of partial provenance data. We then measure the quantitative metrics on these sets of partial provenance data that describe certain important attributes of the reduced data, like size and degree of full provenance and quantify them. They help in comparing how good a reduction technique works in comparison to another based on the size of provenance they collect and the amount of full provenance they hold. When we apply a reduction technique to input data and then execute the Spark program, the result obtained is different from the result obtained from the non-reduced input data. However, the difference in result is ignored as we focus on the provenance data collected. In the next section, we introduce a running example followed by formal description of these quantitative metrics. The running example is used in the description of these metrics to explain them better.

5.4.1 Running example

Consider the running example from Section 4.1 where we present a dataset that contains temperature values across different locations belonging to multiple countries and a Spark program that calculates the average temperature per country. Table 5.9 shows the input data from the example and Table 5.10 shows the result of executing the Spark program P from Section 4.1 on it. The program computes the average temperature per country and hence in the result, we see two tuples, each corresponding to the two countries available in the input data - U.S and Australia. The two tuples t_a and t_b from the result possess full provenance. That is we can track all the tuples in the input data that contributed to each of them. This is because we have collected provenance for all the input tuples. However, this is not the case when we use the provenance data reduction techniques.

t_a	Alaska USA	51.66
t_b	Australia	19.83

Table 5.10: Result of program P executed on non-reduced data

	Date	Latitude	Longitude	Temperature	Country
t_0	26-04-2016	66	-153	51	Alaska USA
t_1	26-04-2016	68	-151	39	Alaska USA
t_2	26-04-2016	65	-152	60	Alaska USA

Table 5.11: Reduced data

We collect partial provenance in order to reduce the size of provenance collected. When we apply the provenance data reduction techniques, we collect provenance only for a subset of input tuples. So we do not have full provenance for all the tuples present in the result of executing a program on reduced data. All the input tuples which would have contributed to the result may not be a part of reduced provenance data and hence are not identified as part of the provenance of the result tuples.

The following example illustrates this. Table 5.11 shows the reduced data obtained after applying some data reduction technique. We run the same program P from Section 4.1 on this reduced data that calculates the average temperature per country. Table 5.12 shows the obtained result.

Table 5.11 shows the tuples for which the provenance is collected and stored. In the case of non-reduced data, provenance was collected and stored for result with two tuples. The data reduction has caused one of the tuples t_b to have no provenance associated with it at all. In addition, $t_{a'}$ possesses partial provenance rather than full provenance, because the set of tuples that contribute to $t_{a'}$ in the case of reduced data is t_0,t_1 and t_2. The tuples contributing to t_a are t_0,t_1,t_2,t_5,t_6 and t_7.

The following sections introduce and describe the quantitative metrics:

5.4.2 Metric 1 - Size of provenance

Because the aim of the thesis is to reduce the overall amount of provenance data collected, we consider the size of the collected provenance data as one of the metrics that quantify the effectiveness of a data reduction technique. We measure the size occupied by full provenance

$t_{a'}$	Alaska USA	50.0

Table 5.12: Result of program P executed on data in Table 5.11

data in bytes and then compare it to the size occupied by the partial provenance obtained by each of the reduction techniques.

Consider the running example from Section 5.4.1. We use Titian to collect provenance for the tuples t_a and t_b from the result of Spark program with non reduced input data. We save the provenance as an object file and measure the size of the file. It is 6 kilobytes. We then collect provenance for the tuple $t_{a'}$ which is the result of the program executed on the reduced data using Titian. Saving the provenance associated with this tuple as object file, we measure and find it to be 1.8 kilobytes. We see that the size of provenance is close to the size of data reduced.

5.4.3 Metric 2 - Number of result tuples with full, partial and no provenance

When we reduce provenance data, there is a possibility that few tuples in the result may not have provenance associated with them at all. Also, there may be certain tuples in the result that either have full provenance or partial provenance associated. A technique that produces reduced provenance where most of the tuples have partial provenance associated with them is better than a technique that produces reduced provenance where most of the tuples have no provenance associated with them. Hence, for every reduced provenance data obtained from different techniques, we measure the number of tuples in the result of a program that have full provenance, partial provenance and no provenance. These values help us compare what reduced provenance data holds more provenance.

Let *I* be the non-reduced input data, and let $M = |I|$ be the number of tuples in *I*. Let *Res* be the set of result tuples obtained by executing a program on non-reduced input data, and let $N = |Res|$ be the number of tuples in the *Res*. Let *I'* be the reduced data, and let $M' = |I'|$ be the number of tuples in *I'*. Let *Res'* be the set of result tuples obtained by executing a program on non-reduced input data, and let $N' = |Res'|$ be the number of tuples in the *Res'*. Then, we have, in case of monotonic workflows,

$$M' \leq M \implies N' \leq N$$

where $M, M', N, N' \in \mathbb{N}$

The difference *N - N'* gives the number of tuples in the result with no provenance. That is, N'_{no}=*N - N'*. Let prov(*t*) be the set of tuples from input data that contributed to *t*. Let prov'(*t*) be the set of tuples from reduced data that contributed to *t*. A tuple $t \in$ *Res'* possesses full provenance if and only if $|prov(t)| = |prov'(t)|$. It possesses partial provenance if $|prov(t)| < |prov'(t)|$ and it possesses no provenance if $|prov'(t)| = 0$. The number of result tuples with full, partial and no provenance add up to the all tuples in the result set.

$$N'_{result} = N'_{full} + N'_{partial} + N'_{no}$$

where $N'_{full}, N'_{partial}, N'_{no}$ are the number of tuples $\in$ *Res'* that possess full provenance, partial provenance and no provenance respectively.

Consider the running example from Section 5.4.1. Both the tuples in the result of non-reduced data have full provenance. Hence in that case, $N'_{full} = 2$, $N'_{partial} = 0$ and $N'_{no} = 0$. In the case of reduced provenance data, the tuple $t_{a'}$ does not have full provenance. That is because, in the case of non-reduced provenance, the number of tuples in input t_a depends on is 6. However, in case of reduced provenance, $t_{a'}$ depends only on 3 tuples. $t_{a'}$ possesses partial provenance. The tuple $t_{b'}$ does not have any provenance associated with it in reduced provenance. Hence, in this reduced provenance data, $N'_{full} = 0$, $N'_{partial} = 1$ and $N'_{no} = 1$.

5.4.4 Metric 3 - Partial provenance metric

The previous metric provides the number of tuples in the result of a program executed on reduced data that contain full provenance N'_{full}, partial provenance $N'_{partial}$ and no provenance N'_{no}. This metric quantifies the degree of full provenance present in the result of a program executed on reduced data. That is, partial provenance metric specifies the amount of full provenance possessed by the set of tuples that belong to *Res'*.

prov(*t*) is the set of tuples from input data that contributed to *t* and prov'(*t*) is the set of tuples from reduced data that contributed to *t* as seen in previous section. When we compute $|prov(t)|$ for all the tuples in *Res* and add them, we get the total number of tuples in non-reduced input data that have contributed to the result tuples in *Res*, represented by $\sum_{i=0}^{N} |prov(t)|$. The difference $|prov(t)| - |prov'(t)|$ quantifies the amount of provenance lost for every tuple in the result of reduced data due to provenance data reduction. We then add this difference for all the result tuples in *Res* to compute the overall amount of provenance lost by the entire result set *Res'*. This summation of the difference is represented as $\sum_{i=0}^{N}(|prov(t)| - |prov'(t)|)$. The full provenance possessed by the set of tuples in *Res* is obtained by adding provenance of all the tuples in the result set *Res* as $\sum_{i=0}^{N} |prov(t)|$.

To calculate the overall amount of full provenance possessed by the complete result set of a program on reduced data, we measure the partial provenance metric (PPM) in the following way:

$$PPM = 1 - \frac{\sum_{i=0}^{N}(|prov(t)| - |prov'(t)|)}{\sum_{i=0}^{N} |prov(t)|}$$

The PPM of a result set of a program lies between 0 and 1. The larger the value of PPM, the more complete is the amount of provenance possessed by the result set. When all the tuples in the result have full provenance, the PPM of the result set is 1. In no provenance exists, the PPM is 0.

Let us have a look at our running example. The number of tuples in the reduced result set that possess full provenance is 0, that possess partial provenance is 1 and that possess no provenance is 1, that is $N'_{full} = 0$, $N'_{partial} = 1$ and $N'_{no} = 1$. The original result contains two tuples. Therefore, *N* = 2 and *N'* = 1.

To calculate the PPM for *Res'*, we must calculate the $|prov(t)| - |prov'(t)|$ for every tuple $t_i \in$ *Res* with its corresponding tuple $t_j \in$ *Res'*.

Res = $\{t_a, t_b\}$

Res' = $\{t_{a'}\}$

For tuple t_a, $|prov(t_a)| - |prov'(t_{a'})| = |\{t_0, t_1, t_2, t_5, t_6, t_7\}| - |\{t_0, t_1, t_2\}|$ = 6 - 3 = 3

For tuple t_b, $|prov(t_b)| - |prov'(t_{b'})|$ = $|\{t_0, t_1, t_2, t_5, t_6, t_7\}|$ - $|\{\}|$ = 6

$\sum_{i=0}^{N} |prov(t_i)|$ = Number of tuples in input that contributed to t_a + Number of tuples in input that contributed to t_b

Substituting the values above in

$$PPM = 1 - \frac{\sum_{i=0}^{N}(|prov(t_i)| - |prov'(t_i)|)}{\sum_{i=0}^{N} |prov(t_i)|}$$

$$=1 - \frac{6+3}{12}$$

$$=0.25$$

Hence tuples in Res' possess 25% of full provenance.

For every scenario introduced in Section 5.1, we apply the 4 provenance data reduction techniques in order to obtain the respective reduced provenance data. We execute the scenarios described in previous section on these reduced sets of data. Then we compute the quantitative metrics listed above for the results of the scenarios executed on each of the reduced sets of data. These metrics allow us to compare and contrast the provenance data reduction achieved by utilizing the four techniques. They help assess the effectiveness of each of these techniques with respect to the scenario involved. We collectively consider all three metrics to conclude what technique works best for a scenario.

5.4.5 Evaluation results

In this section, we apply the previously defined metrics on the obtained provenance from each scenario from Section 5.1. We measure the metrics on provenance data of the results obtained from executing the spark programs with input file sizes of 100MB, 500MB, 1GB and 1.5GB (in case of radiation dataset). The metrics are found to be similar across the input file sizes. Hence, we present only the results obtained from input file size of 1GB. We first present the results on provenance data collected from the Spark programs based on radiation dataset followed by the results obtained from those based on U.S Domestic flights dataset.

5.4.5.1 Results for radiation dataset

We present the results obtained from a quantitative analysis performed on scenarios of the radiation dataset in the following sections:

Scenario 1

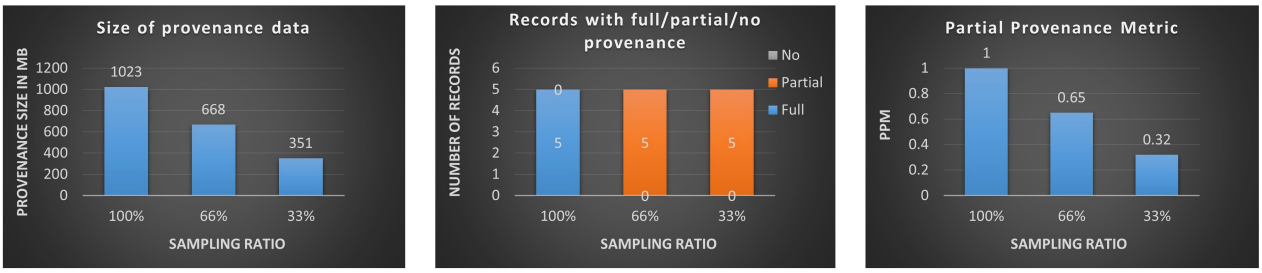
This scenario aims at finding 10 highest radiation readings per year from years 2010 until 2016. We first collect full provenance during the execution of the Spark program and compute quantitative metrics on the full provenance data. The individual metrics computed for full provenance data are shown against the sampling ratio of 100% in Figure 5.1. We then perform four data reduction techniques - random sampling, histogram, clustering and equivalence classes on the result to obtain partial provenance.

Random Sampling

In random sampling, we reduce the provenance data based on the sampling ratio. We choose sampling ratios - 66% and 33% for our evaluation. In Figure 5.1, we present the quantitative metrics for random sampling with the given sampling ratios. We see that the size of the provenance data collected is linearly proportional to the sampling ratio. The reduced provenance data collects at least partial provenance for all the result tuples. There are no tuples in the result without any provenance as shown in Figure 5.1 b). The partial provenance metric shows that the amount of full provenance present in the tuples of reduced provenance data is close to the sampling ratio of data reduction.

Histogram Analysis

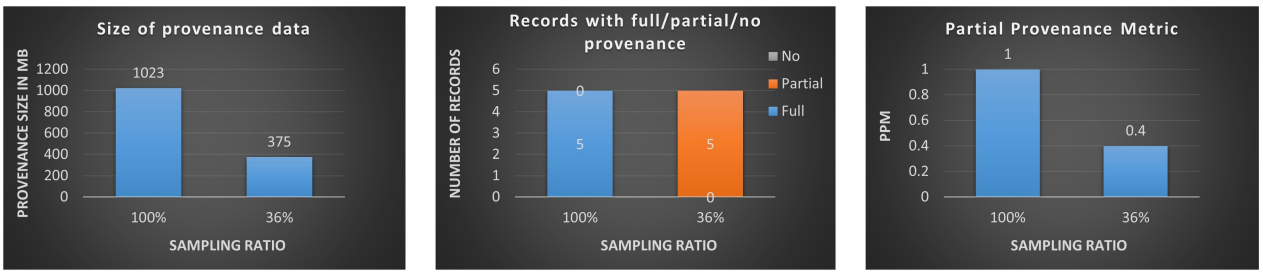
In case of histogram-based provenance data reduction, we collect provenance only for the tuples that have the most frequently occurring value for radiation. We obtain frequency distribution for values of radiation level and select the range of values that occur most frequently. Figure 5.2 shows the metrics computed on the histogram-based reduced provenance data. We find that the proportion of such tuples is 35% in our data. The size of provenance data is close to 35% which



(a) (b) (c)

Figure 5.1: Quantitative metrics for result of executing Spark program of Scenario 1 on data reduced by random sampling

is the sampling ratio. The partial provenance metric shows that the amount of full provenance possessed by the tuples in the reduced provenance data is 40%. This is more than the sampling ratio of 35%.



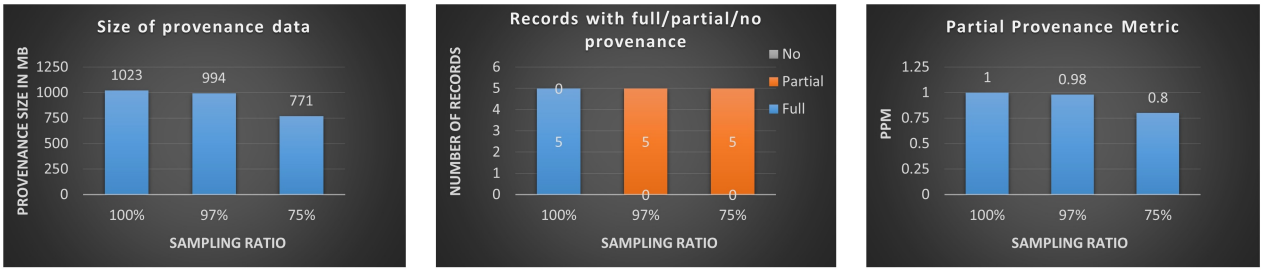
(a) (b) (c)

Figure 5.2: Quantitative metrics for result of executing Spark program of Scenario 1 on data reduced by histogram analysis

Clustering

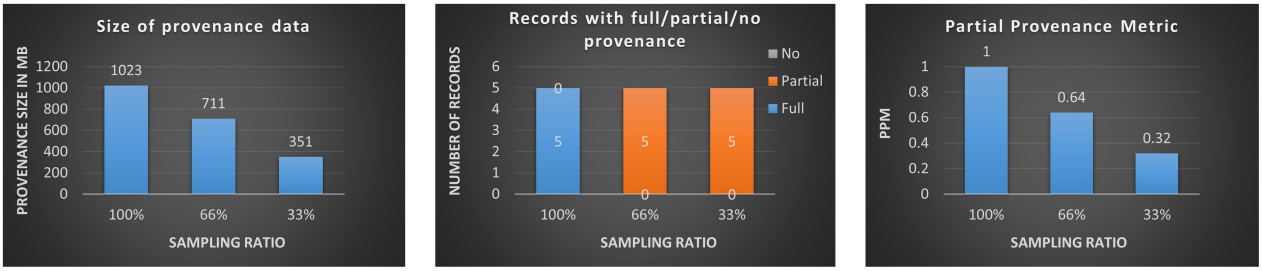
In case of clustering-based provenance data reduction, we can reduce the provenance in two ways - form clusters within input data and retain provenance only for tuples that belong to certain clusters, and form clusters within input data and retain provenance for a certain set of tuples that belong to each cluster. Figure 5.3 shows the results. We randomly select few clusters and retain provenance for the tuples that belong only to these few clusters. Even in this case, the size of the provenance is linearly proportional to the size of input data. However, the amount of full provenance after provenance data reduction of 75% is 80%. Figure 5.4 presents the metrics obtained in the case of sampling the tuples within every cluster. From every cluster formed in the input data, we select 66% and 33% of the tuples and collect provenance only for them. In Figure 5.4 (a), we see that the size of provenance data collected is linearly proportional to the size of the clusters after randomly eliminating few tuples. We also see that there are no tuples

in the result without any provenance from Figure 5.4 (b). The sampling ratio of the reduced provenance data and the amount of full provenance present in it are closely related. In this case, the sampling ratio is equal to the PPM. Figure 5.4 (c) illustrates this.



(a) (b) (c)

Figure 5.3: Quantitative metrics for result of executing Spark program of Scenario 1 on data reduced by randomly eliminating 66% and 33% of the clusters



(a) (b) (c)

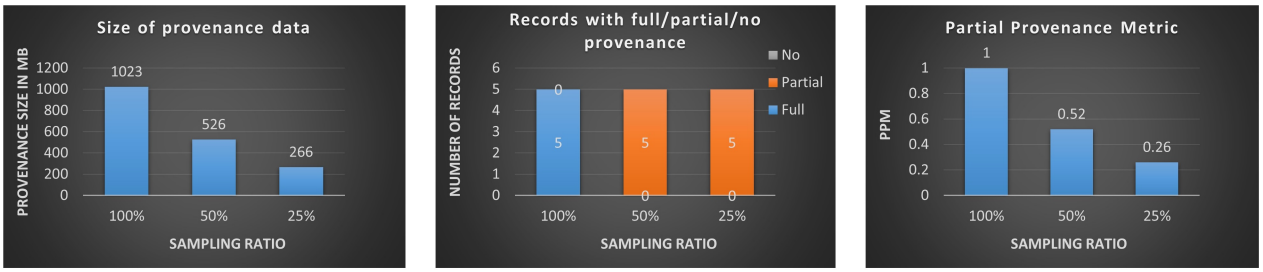
Figure 5.4: Quantitative metrics for result of executing Spark program of Scenario 1 on data reduced by eliminating records from clusters

Equivalence classes

Now we look at the equivalence class based reduction. We can reduce provenance data in two ways in case of equivalence class based data reduction. We first form equivalence classes within the input data and then either collect provenance for tuples belonging to only a few equivalence classes, or collect provenance for tuples randomly selected from every equivalence class. In the former case, we form equivalence classes based on the attribute "year" of the tuples and eliminate tuples that belong to few years. Figure 5.5 shows the metrics computed on the reduced provenance data obtained by eliminating all tuples belonging to certain randomly selected years. Figure 5.5 (a) shows that the size of the provenance data is linearly proportional to the sampling ratio of the reduced provenance data. Also, there are no tuples in the result with full provenance, nor are there any tuples that have no provenance associated with them. The result contains 5 tuples which all possess partial provenance. Figure 5.5 (c) shows that the partial provenance

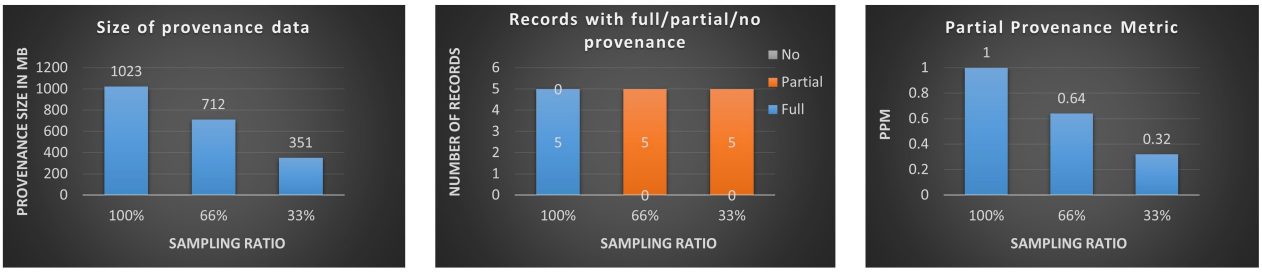
metric is very close to the size of the reduced provenance data. For instance, the result tuples in 33% of reduced provenance data possesses 32% of full provenance.

In the second way of reducing provenance data by applying equivalence classes, we first form equivalence classes on the input data based on the attribute "year" of the tuples and collect provenance for only 66% and 33% of the classes formed. We collect 66% and 33% of tuples that belong to each year. After eliminating certain classes corresponding to the attribute "year" from the provenance, we obtain reduced provenance data of size 50% and 25%. Figure 5.6 (a) shows that size occupied by the provenance data is linearly proportional to the size of the reduced provenance data. Also, the result contains no tuples without any provenance and that provenance data reduction has caused all the tuples to have partial provenance. Figure5.6 (c) illustrates the partial provenance metric that shows that the amount of full provenance present in the reduced provenance data is equal to its sampling ratio like in the other cases.



(a) (b) (c)

Figure 5.5: Quantitative metrics for result of executing Spark program of Scenario 1 on data reduced by randomly eliminating 66% and 33% of the equivalence classes



(a) (b) (c)

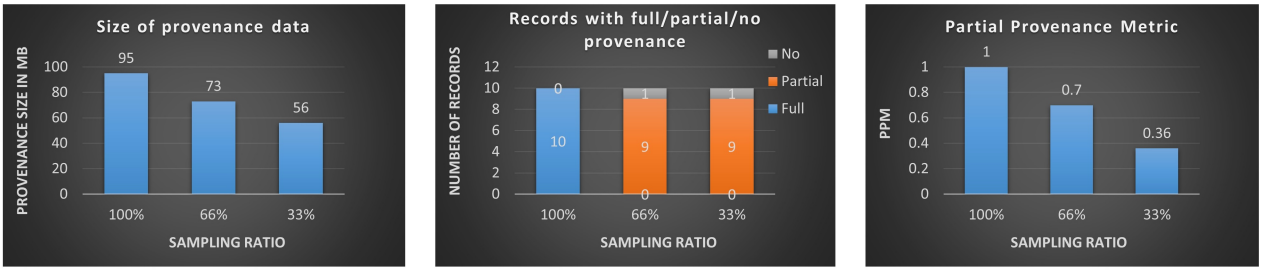
Figure 5.6: Quantitative metrics for result of executing Spark program of Scenario 1 on data reduced by eliminating records from equivalence classes

Scenario 2

The Spark program as part of this scenario finds the average radiation per year and month in the U.S. We first collect full provenance using Titian during the execution of the program. The result tuples obtained possess information on all the tuples from the input data that contributed to them.

Random Sampling

We then perform random sampling on input data and then select only 66% and 33% of the tuples from the input. We then use Titian to collect provenance for only these tuples. The result tuples in this case do not always possess full provenance as few of the input tuples that contribute to them are sampled out and do not have provenance associated with them. Figure 5.7 (a) shows the size occupied by the reduced provenance data obtained by randomly sampling 66% and 33% of provenance data and collecting provenance only for them. The size of the provenance data is worse than linearly proportional to the sampling ratio as was in the previous scenario. That is because, the Spark program executed contains two reduce operations. We perform sampling only on the input data, however we do not sample the tuples after the first reduce operation. Figure 5.7 (b) shows that in the provenance data sample of 66% we have a tuples in the result without any provenance. Even when we collect provenance for only 33% of input tuples, we have only one tuple that has no provenance. Rest of the nine tuples have partial provenance associated with them. Figure 5.7 (c) shows the partial provenance metric of the reduced provenance data. We see that in case of sampling ratio of 66%, the PPM is 0.7. That is the even though we are collecting provenance for only 66% of the input, the tuples in the result possess 70% of full provenance. This is because, random sampling eliminated the input tuples that contributed to the one of the result tuples completely (the one result tuple without any provenance), while retaining 70% of input tuples that correspond to the 9 result tuples which possess partial provenance.

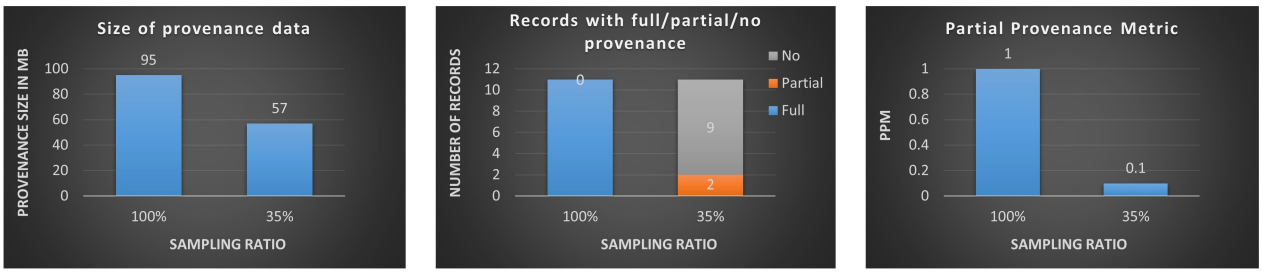


(a) (b) (c)

Figure 5.7: Quantitative metrics for result of executing Spark program of Scenario 2 on data reduced by random sampling

Histogram analysis

In case of histogram-based provenance data reduction, we reduce the provenance data in two ways - we form histogram of radiation values and collect provenance for the set of tuples from input that belong to the range of frequently occurring values, and we form histogram on the number of input tuples a result tuple is dependent on and collect provenance for the result tuple that has most number of input tuples that contributed to it. Figure 5.8 (a) shows the size occupied by the non-reduced and reduced provenance data. The frequently occurring radiation values correspond to 35% of the input data and we see that the provenance data size is not linearly proportional to the sampling ratio as was the case with random sampling. Figure 5.8 (b) shows that the number of tuples without any provenance associated with it is large in this case, i.e., 9. We collect partial provenance for only 2 tuples out of 11 tuples. Figure 5.8 (c) shows that the partial provenance metric of the result set corresponding to reduced provenance data is 0.1. To conclude, this approach does not work well.



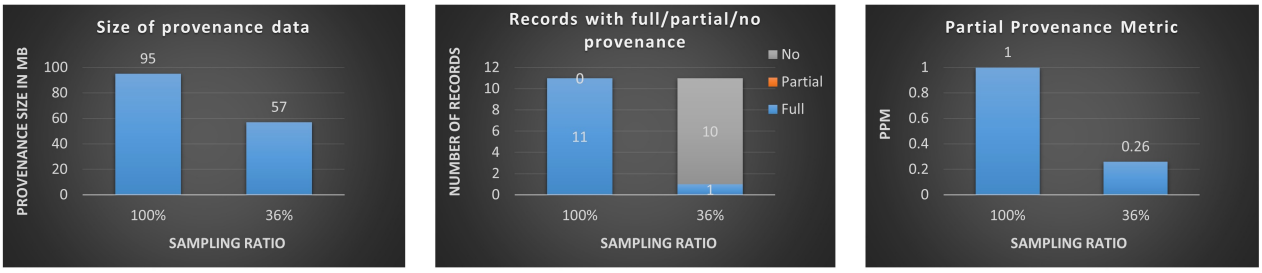
(a) (b) (c)

Figure 5.8: Quantitative metrics for result of executing Spark program of Scenario 2 on data reduced by histogram analysis

In the second histogram-based provenance data reduction, we collect provenance for only one tuple in the result that has the highest number of input tuples that contributed to it. Figure 5.9 (a) shows that the size of provenance data is not linearly proportional to the number of the input tuples that contributed to the one tuple in the result which we have considered. Figure 5.9 (b) shows that we have one tuple in the result that has full provenance. However, the result of 10 tuples do not have any provenance associated with them at all. Figure 5.9 (c) shows that PPM of the reduced provenance is 0.36 which is close to the sampling ratio. In this case, it reflects the number of input tuples that contributed to the tuple which has full provenance.

Clustering

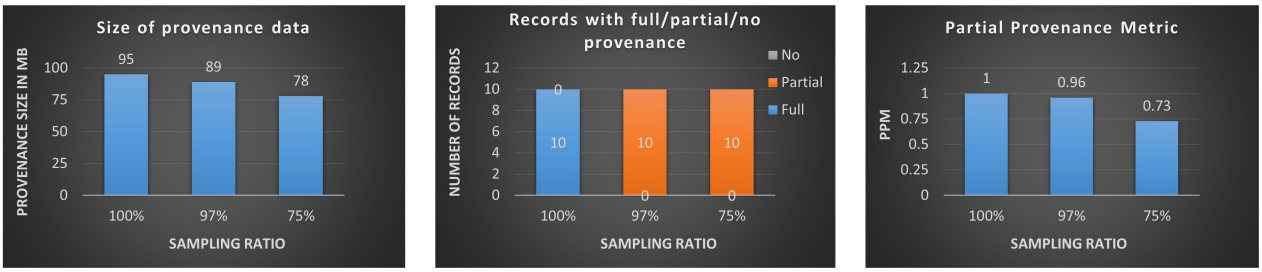
As in case of Scenario 1, we reduce provenance using clustering in two ways - by eliminating complete clusters formed in input data and collecting provenance for rest of them, or randomly selecting tuples from every cluster and collecting provenance for them. Figure 5.10 shows the



(a) (b) (c)

Figure 5.9: Quantitative metrics for result of executing Spark program of Scenario 2 on data reduced by histogram analysis by collecting provenance for the tuple that is dependent on most input tuples

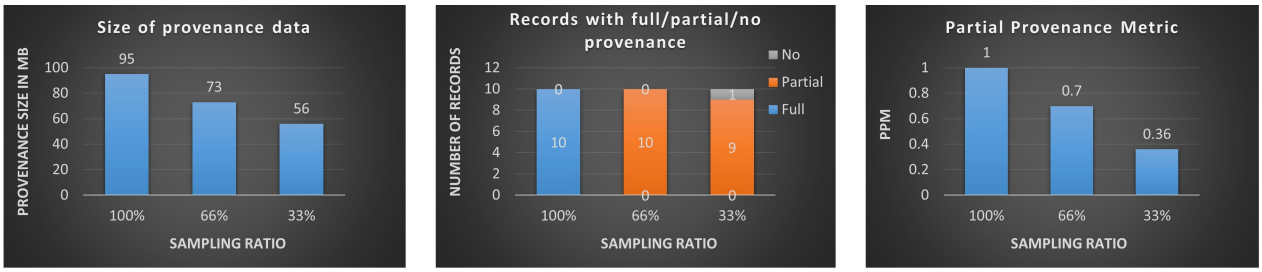
metric of the case where we collect provenance for input tuples within complete clusters. The size of provenance data collected is not linearly proportional to the number of tuples present in the clusters that have been considered for provenance collection, i.e., the sampling ratio. The reason being the presence of two reduce operators in the Spark program. Figure 5.10 (b) shows there are no tuples in the result without any provenance associated with it. The PPM is also proportional to the size of the reduced provenance data as was the case in Clustering scenarios in Scenario 1.



(a) (b) (c)

Figure 5.10: Quantitative metrics for result of executing Spark program of Scenario 2 on data reduced by randomly eliminating 66% and 33% of the clusters

Figure 5.11 shows the metrics for clustering based provenance data reduction where we collect provenance for randomly selected tuples from every cluster formed within input data. As in the case of the previous clustering approach, the size of the provenance data is not linearly proportional to the sampling ratio, and PPM are dependent on the sampling ratio, i.e., the amount of tuples from input data that were considered for provenance collection. Figure 5.11 (b) however, shows that, in this case we have one tuple in the result that does not have any provenance with it. Rest of the nine tuples from the result possess partial provenance.

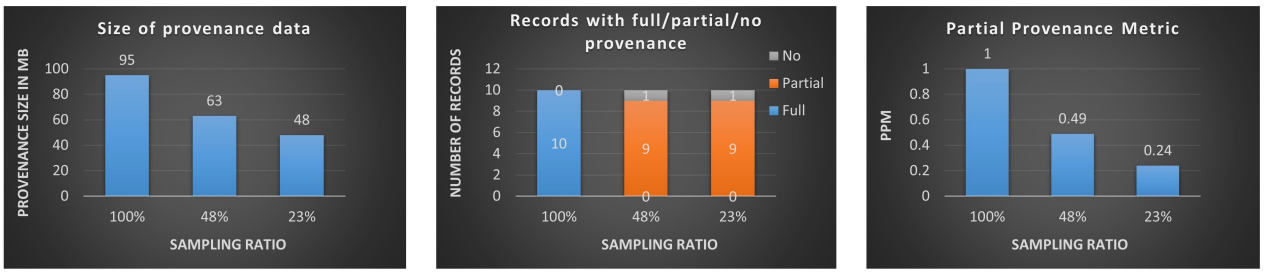


(a) (b) (c)

Figure 5.11: Quantitative metrics for result of executing Spark program of Scenario 2 on data reduced by eliminating records from clusters

Equivalence classes

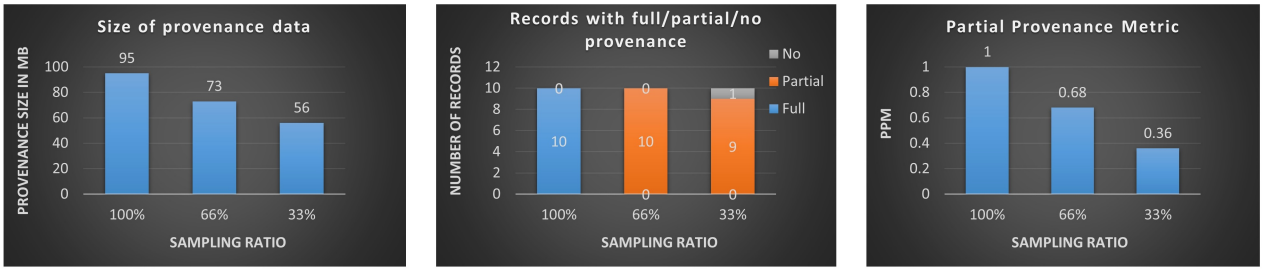
We first form equivalence classes in the input data based on the attribute "year". Then collect provenance in the two introduced ways. We collect provenance for tuples belonging to certain classes, and we collect provenance for certain number of tuples belonging to every class. Figure 5.12 shows the results for the equivalence class based approach that collects provenance for only a few classes, i.e., only a few years. Like in the case of other provenance data reduction techniques, the reduced provenance data size is not linearly proportional to the sampling ratio, or the size of the classes for which provenance was collected. The PPM as shown in Figure 5.12 (c) is close to the sampling ratio. That is, we collect provenance for 48% of the input tuples and the amount of full provenance possessed by them is close to 48%. We have one tuple in the result that does not have any provenance associated with it. The rest of the nine tuples possess partial provenance.



(a) (b) (c)

Figure 5.12: Quantitative metrics for result of executing Spark program of Scenario 2 on data reduced by randomly eliminating 66% and 33% of the equivalence classes

In the case in which we randomly select few input tuples from each class and collect provenance for them, we see that the results look similar to the earlier case. Figure 5.13 shows these results.



(a) (b) (c)

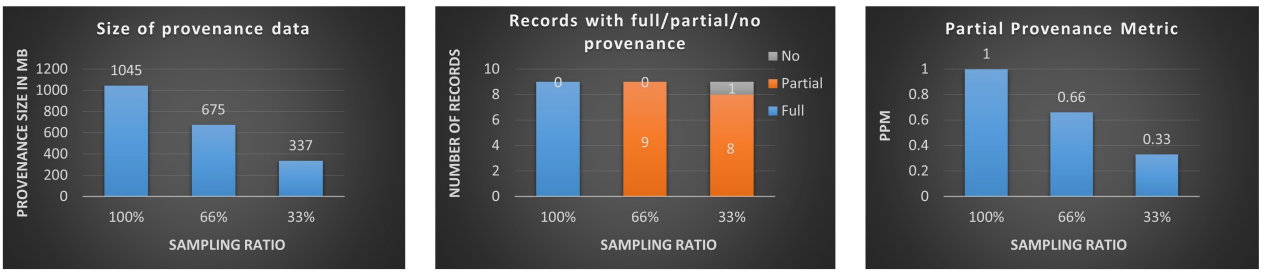
Figure 5.13: Quantitative metrics for result of executing Spark program of Scenario 2 on data reduced by eliminating records from equivalence classes

Scenario 3

In scenario 3, we find the average radiation reading per unit type.

Random sampling

We first apply the random sampling based provenance data reduction technique where we randomly select the input tuples and collect provenance only for them. Figure 5.14 (a) shows the size of the provenance data collected in this case, which is linearly proportional to the number of tuples in input we collect provenance for. Figure 5.14 (b) illustrates that in case of provenance collection of 66% of input tuples, there are no tuples in the result that have no provenance associated with it. However, when we reduce the sampling ratio to 33%, then we see that there is one tuple in the result without any provenance. The rest of the tuples in the result have partial provenance associated with them.

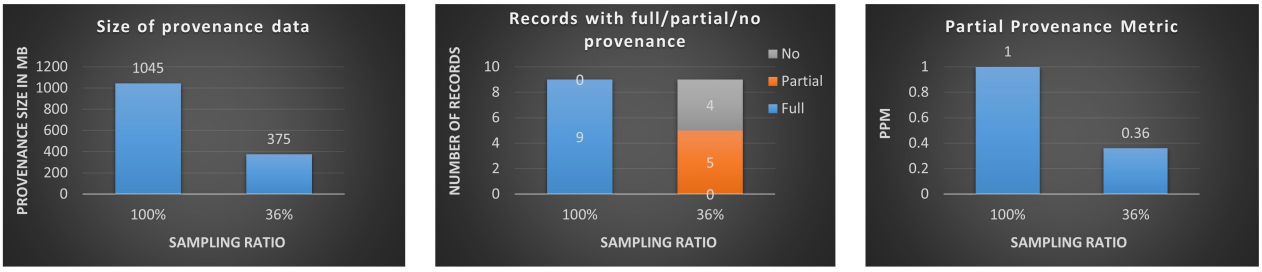


(a) (b) (c)

Figure 5.14: Quantitative metrics for result of executing Spark program of Scenario 3 on data reduced by random sampling

Histogram analysis

In the case of histogram based provenance data reduction, we compute histogram based on the radiation values of the tuples as in the case of scenario 1 and scenario 2, and then collect provenance only for the input tuples that have radiation values in the range of frequently occurring values. There are 36% of input tuples whose radiation value falls in the range of frequently occurring radiation values. Hence the sampling ratio is 36%. Figure 5.15 (a) and (c) show that the size of the provenance data and the partial provenance metric are linearly proportional to the sampling ratio. Figure 5.15 (b) shows that the number of tuples in the result with no provenance is 4. The rest of the 5 tuples have partial provenance. We can infer that the four tuples with no provenance are dependent on the input tuples whose radiation readings do not fall within the frequently occurring radiation readings and hence have no provenance associated with them.



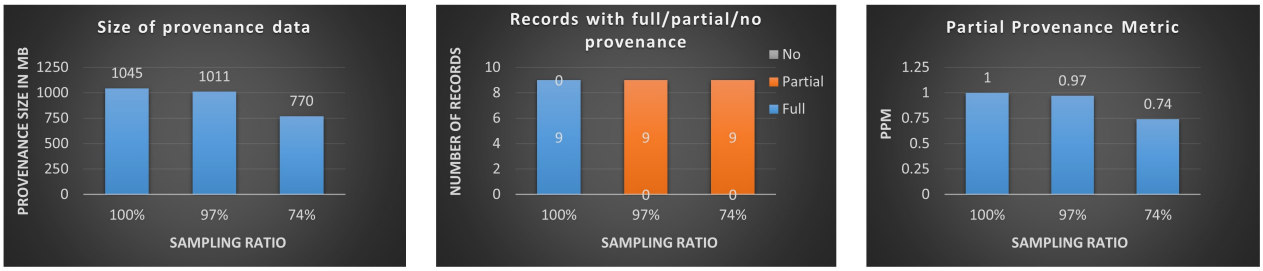
(a) (b) (c)

Figure 5.15: Quantitative metrics for result of executing Spark program of Scenario 3 on data reduced by histogram analysis

Clustering

In case of clustering based provenance data reduction, we form clusters in data using the attributes “latitude” and “longitude”, and then collect provenance for either all tuples from few clusters or for few tuples from all clusters. Figure 5.16 shows the metrics for the former case. We randomly eliminate few clusters and see that the sampling ratios obtained are 97% and 74%. The size of the provenance data is linearly proportional to the sampling ratio. For both the reduced provenance data samples, there are no tuples in the result without any provenance associated with them. The partial provenance metric is close to the sampling ratio. That is collecting provenance for 74% of input tuples provides us with result tuples that have 74% of full provenance.

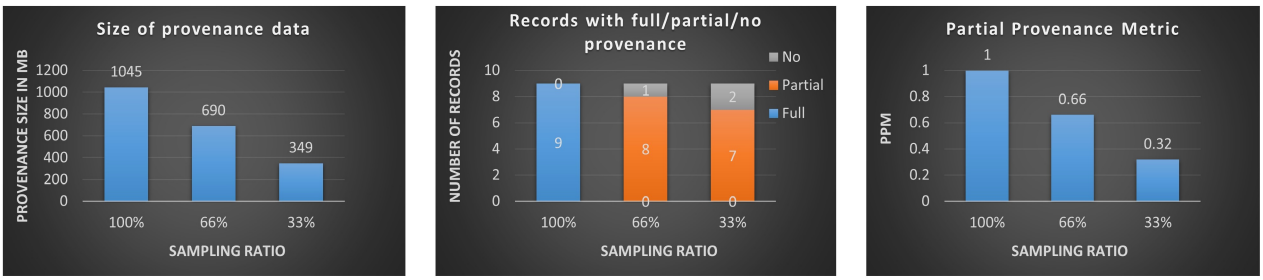
When we collect provenance for 66% and 33% of the tuples belonging to every cluster, we see that we have one tuple and two tuples in the result set with no provenance respectively. Figure



(a) (b) (c)

Figure 5.16: Quantitative metrics for result of executing Spark program of Scenario 3 on data reduced by randomly eliminating few clusters

5.17 (b) illustrates this. In Figure 5.17 (c), we see that the PPM is close to the sampling ratio like in the other cases. Also, the size of the provenance data is linearly proportional to the sampling ratio.

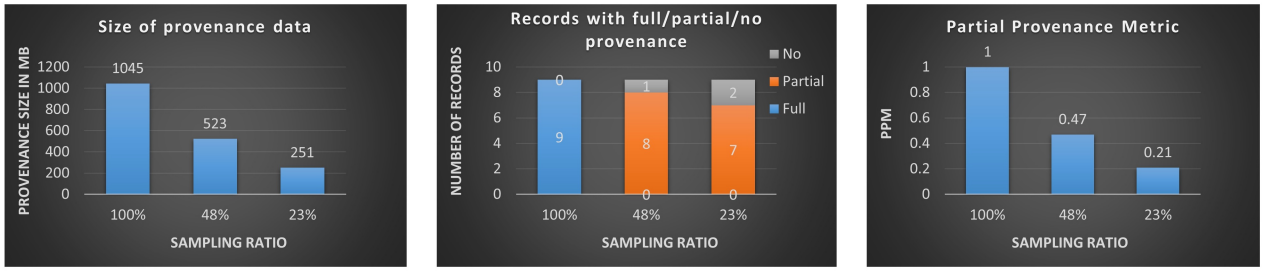


(a) (b) (c)

Figure 5.17: Quantitative metrics for result of executing Spark program of Scenario 3 on data reduced by eliminating records from clusters

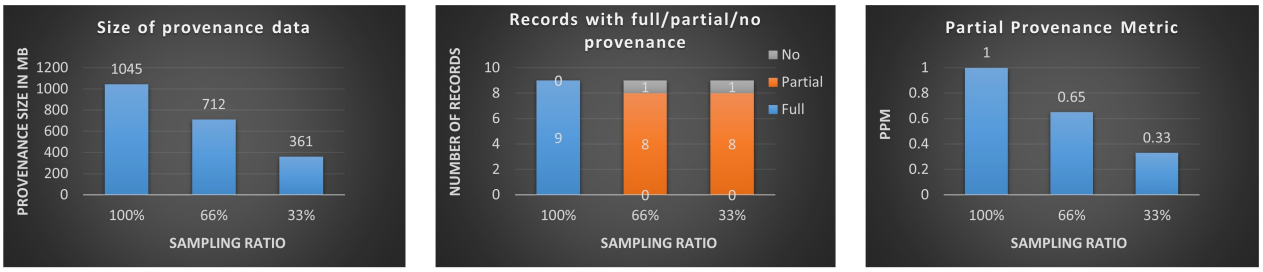
Equivalence classes

In the equivalence class based provenance data reduction, we follow the same two approaches to reduce provenance data as in the previous scenarios. Figure 5.18 and Figure 5.19 show the metrics for reduced provenance data by eliminating classes and eliminating tuples from every class respectively. The size of the provenance data and PPM are linearly proportional to the sampling ratio in both the cases. However, when we collect provenance for all tuples of few classes, which results in a sampling ratio of 23%, there are two tuples with no provenance associated with them. Figure 5.18 (b) illustrates this. In the case of collecting provenance for only few randomly selected tuples from every class, there is only one tuple in result that has no provenance associated with it. Figure 5.19 (b) shows this. The rest of the eight tuples in the result have partial provenance.



(a) (b) (c)

Figure 5.18: Quantitative metrics for result of executing Spark program of Scenario 3 on data reduced by randomly eliminating 66% and 33% equivalence classes



(a) (b) (c)

Figure 5.19: Quantitative metrics for result of executing Spark program of Scenario 3 on data reduced by eliminating records from equivalence classes

5.4.5.2 Results for U.S domestic flights dataset

We present the results obtained from a quantitative analysis performed on scenarios of the U.S domestic flights dataset in the following sections:

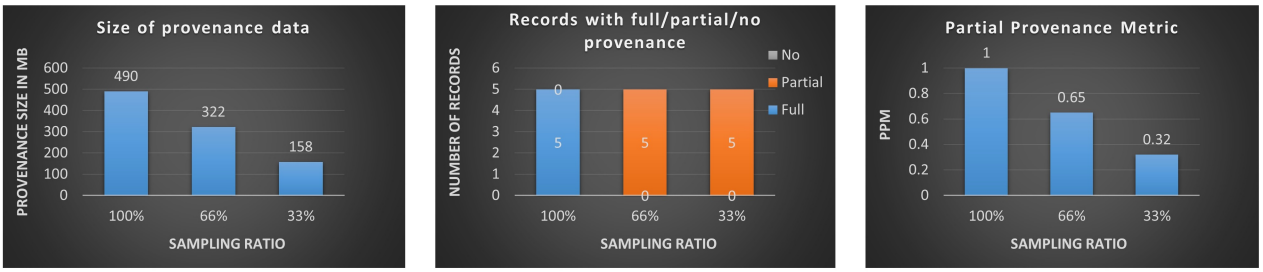
Scenario 4

The Spark program from this scenario finds the average arrival delay for every airline carrier per year. We first collect full provenance using Titian such that the tuples in the result possess information on all the tuples that contributed to them. Then we apply the four provenance data reduction techniques to the input data and create subsets of input data. We then collect provenance using Titian only for those subsets of input data. Hence

we reduce the provenance data corresponding to each of the data reduction techniques.

Random sampling

In the random sampling based provenance data reduction, we randomly select certain number of tuples for which provenance is collected. The number of tuples to be selected is driven by the sampling ratio. We choose two sampling ratios 66% and 33%, where we consider 66% and 33% of input tuples for provenance collection. Figure 5.20 (a) shows the size of provenance data when full provenance, reduced provenance of 66% and reduced provenance of 33% are collected. The size of the provenance data is linearly proportional to the sampling ratio. Figure 5.20 (b) shows that for both 66% and 33% reduced provenance data, there are no tuples in the result of the Spark program without any provenance. Figure 5.20 (c) shows that the PPM is close to the sampling ratio.



(a) (b) (c)

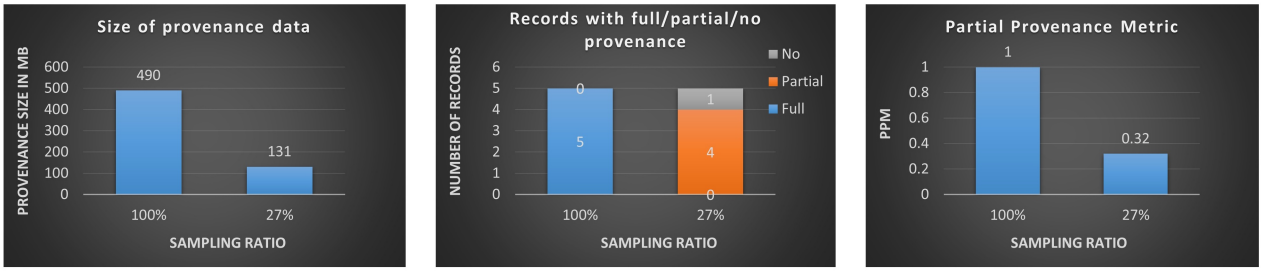
Figure 5.20: Quantitative metrics for result of executing Spark program of Scenario 4 on flight data reduced by random sampling

Histogram analysis

In the histogram-based provenance data reduction approach, we create histogram based on the arrival delay values and select the tuples that fall in the range of most frequently occurring set of values. We collect provenance only for these selected tuples. In our case, this set of tuples in the input accounted for 27% and hence the sampling ratio is 27%. Figure 5.21 (b) shows that there is a tuple in the reduced provenance set which has no provenance associated with it. Figure 5.20 (c) shows the PPM as 0.32 for the sampling ratio of 27%. This is because the data reduction caused one tuple in the result and its contributing tuples to be completely eliminated, whereas the amount of contributing tuples to the rest of the result tuples is close to 32%.

Clustering

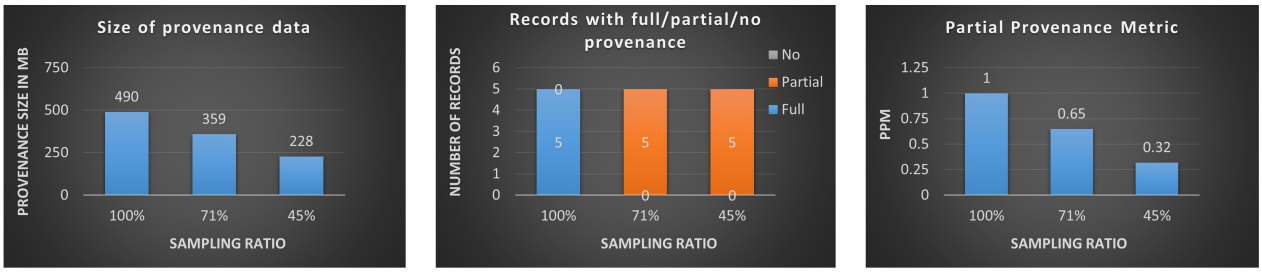
In the clustering-based provenance data reduction approach, we create clusters within the data based on the attributes “year” and “day of the week”. We then collect provenance for either



(a) (b) (c)

Figure 5.21: Quantitative metrics for result of executing Spark program of Scenario 4 on flight data reduced by histogram analysis

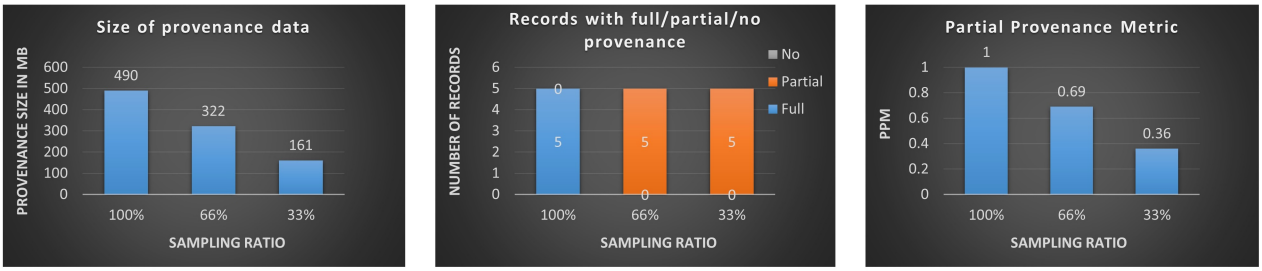
all tuples belonging to few clusters, or few tuples belonging to all clusters. In the former case, we eliminate complete clusters and collected provenance for few randomly selected clusters. The number of tuples present in such clusters amounts to 71% and 45% of the number of input tuples, resulting in sampling ratios of 71% and 45%. Figure 5.22 (a) and Figure 5.22 (c) show that the size of the provenance data and the PPM is linearly proportional to the sampling ratio. Figure 5.22 (b) shows that there are no tuples in both 71% and 45% of reduced provenance data without any provenance.



(a) (b) (c)

Figure 5.22: Quantitative metrics for result of executing Spark program of Scenario 4 on flight data reduced by randomly eliminating 66% and 33% of the clusters

In the latter case of collecting provenance for few tuples from every cluster, we choose a sampling ratio of 66% and 33% and select 66% and 33% of the input tuples for provenance collection. Figure 5.22 (a) and Figure 5.22 (c) show that the size of the provenance and the PPM are linearly proportional to the sampling ratio. Also, there are no tuples in the result that have no provenance associated with them. The metrics are similar to the metrics obtained with the other approach of clustering where we collected provenance for all tuples belonging to few clusters.

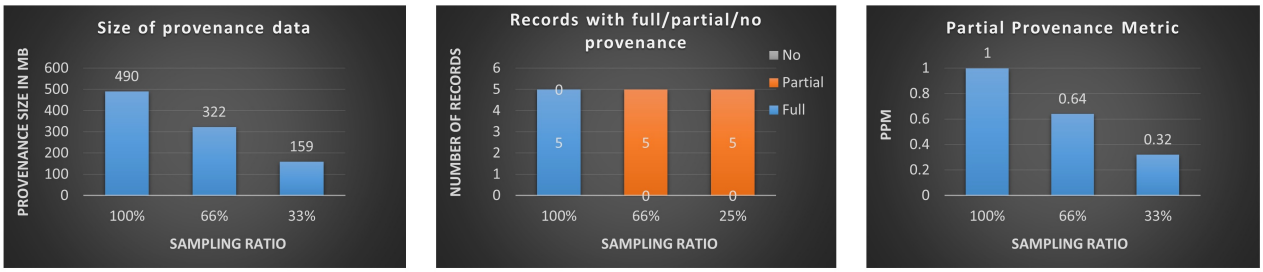


(a) (b) (c)

Figure 5.23: Quantitative metrics for result of executing Spark program of Scenario 1 on flight data reduced by eliminating records from clusters

Equivalence classes

In the equivalence class based approach of provenance data reduction, we collect partial provenance in two ways. We first form equivalence classes on the input data based on the attribute "year", then, we either collect provenance for all the input tuples in few classes or we collect provenance for few tuples in every class. Figure 5.24 shows the metrics obtained when we collect provenance for all tuples from few classes. We eliminate few classes and collect provenance for 66% and 33% of the tuples in input. The size of the provenance is linearly proportional to the sampling ratio. Also Figure 5.24 (b) shows that the PPM is close to sampling ratio. In this case, we do not have any tuple in the result without any provenance.

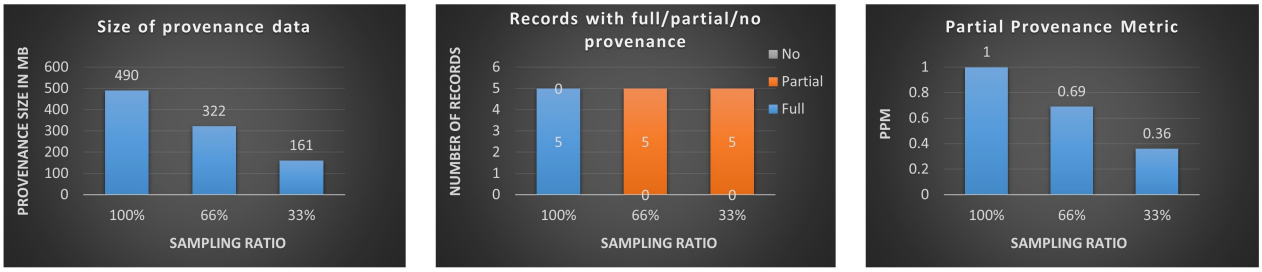


(a) (b) (c)

Figure 5.24: Quantitative metrics for result of executing Spark program of Scenario 4 on flight data reduced by randomly eliminating 66% and 33% of the equivalence classes

Figure 5.25 shows the metrics obtained by collecting provenance for 66% and 33% of tuples belonging to every class,i.e., we randomly sample 66% and 33% of tuples belonging to every year within the input data. The results obtained are the same as in the earlier case, except for the PPM in case of 33% of sampling ratio. The PPM is more in this case, meaning that 33% of the reduced provenance data holds 36% of full provenance. Hence eliminating few tuples from

every class works better than eliminating all tuples from few classes in this scenario.



(a) (b) (c)

Figure 5.25: Quantitative metrics for result of executing Spark program of Scenario 4 on data reduced by eliminating records from equivalence classes

Scenario 5

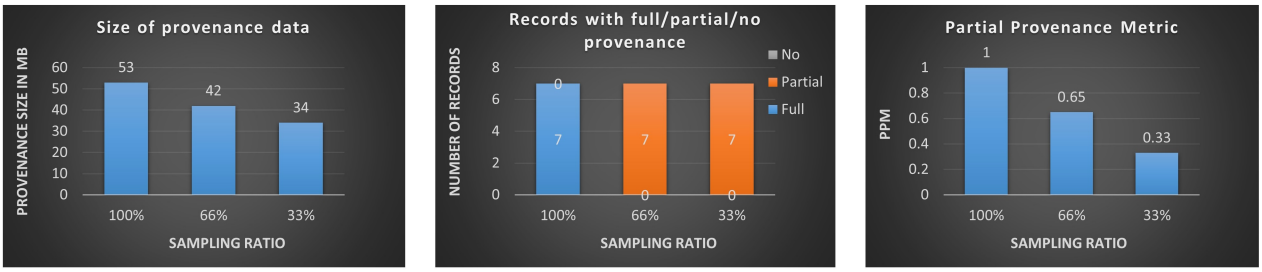
In this scenario, we find the average arrival delay in minutes based on the day of the week. We first execute the Spark program from the scenario and collect full provenance for the tuples in its result. We then apply the data reduction techniques on the input data and collect partial provenance for the reduced input as was the case with other scenarios.

Random sampling

We first apply random sampling on the input data with sampling ratio of 66% and 33% and obtain input data samples of sizes 66% and 33% respectively. We then collect provenance for only these samples of input data. We compute the quantitative metrics on this reduced provenance data and compare them with the metrics computed on the full provenance data. Figure 5.26 (a) shows the size of the provenance data. The size of the provenance data is not linearly proportional to the sampling ratio as was in the previous scenario. That is because, the Spark program executed contains two reduce operations. We perform sampling only on the input data, however we do not sample the tuples after the first reduce operation. Figure 5.26 (b) shows that there are no tuples in the result in case of sampling ratio of both 66% and 33% without any provenance. The PPM is close to the sampling ratio as illustrated by Figure 5.26.

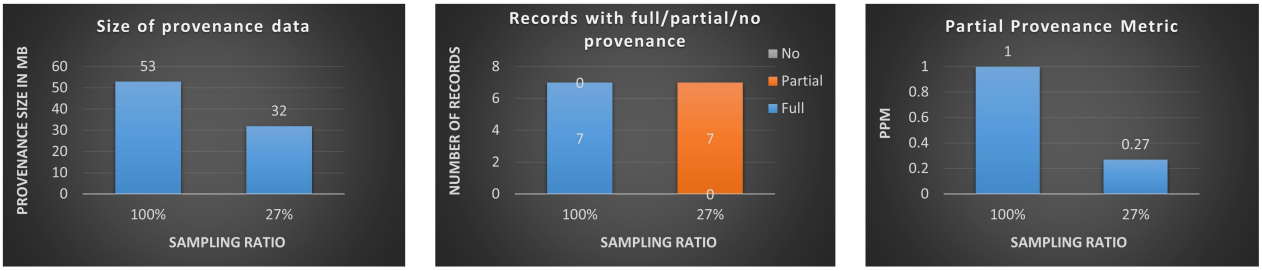
Histogram analysis

In the case of histogram-based provenance data reduction approach, we create histogram and collect provenance for reduced data in the same way as in the case of Scenario 4. Figure 5.27 shows the results obtained in this case. They show that there are no tuples without any provenance in the reduced provenance and that the PPM is close to the sampling ratio.



(a) (b) (c)

Figure 5.26: Quantitative metrics for result of executing Spark program of Scenario 5 on data reduced by random sampling



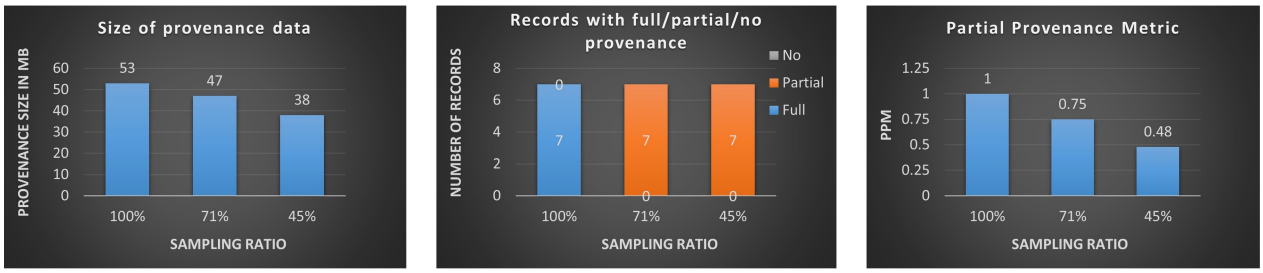
(a) (b) (c)

Figure 5.27: Quantitative metrics for result of executing Spark program of Scenario 5 on data reduced by histogram analysis

Clustering

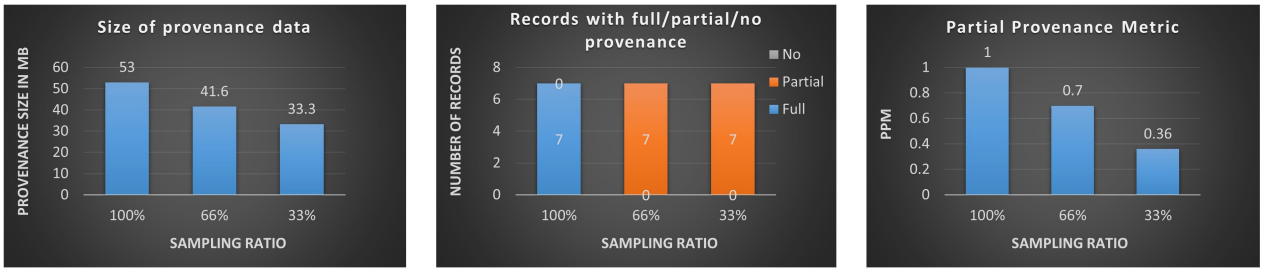
In the clustering based approach of provenance data reduction, we first form clusters in data in the same way we did in the other scenarios and then collect provenance for either all tuples of few clusters or collect provenance for few tuples from all clusters. Figure 5.28 shows the metrics obtained in the case of collecting provenance of all tuples of few clusters. We eliminate few clusters and collect provenance for all tuples of rest of the clusters, which accounts for 71% and 45% of the input data. These are the sampling ratios. The size of the provenance data is proportional to the sampling ratio, however, it is not linearly proportional as was the case in Scenario 4. This is due to the two reduce operators present in the Spark program of this scenario. Figure 5.28 (b) shows that there are no tuples in the reduced provenance data with sampling ratios of 71% and 45% without any provenance. Figure 5.28 (c) shows that the PPM for the reduced provenance data is close to the sampling ratio.

The metrics obtained from provenance data of the clustering-based provenance data reduction in which we collect provenance for few tuples from all clusters are similar to those of random sampling. Figure 5.29 shows these results.



(a) (b) (c)

Figure 5.28: Quantitative metrics for result of executing Spark program of Scenario 5 on data reduced by randomly eliminating 66% and 33% of the clusters



(a) (b) (c)

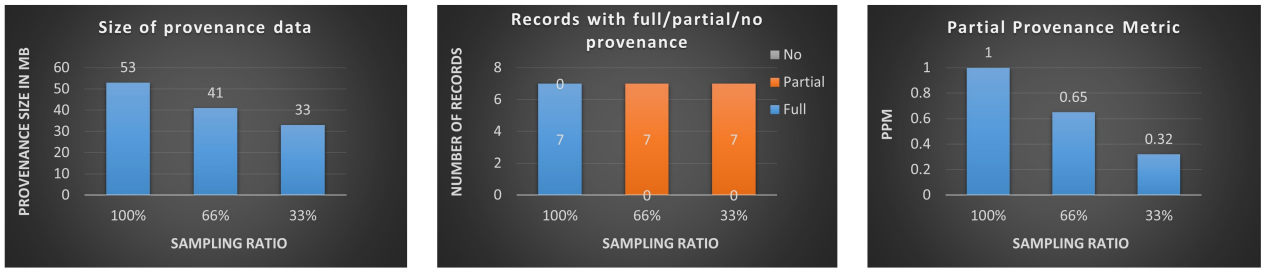
Figure 5.29: Quantitative metrics for result of executing Spark program of Scenario 5 on data reduced by eliminating records from clusters

Equivalence classes

In the equivalence-class based approach of provenance data reduction, we first form equivalence classes within the input data then reduce data in two ways, as in the case of Scenario 4. Figure 5.30 shows the results obtained by collecting provenance only for tuples belonging to few classes based on attribute “year”. We see that the results look similar to that of the random sampling approach. Figure 5.31 show the metrics obtained by collecting provenance for few tuples belonging to every cluster. In Figure 5.31 (b), we see that there is one tuple in the result that has no provenance associated with it. This is the case for both the reduced provenance data with sampling ratio of 66% and 33%. Figure 5.31 (c) shows the PPM corresponding to the sampling ratios of 66% and 33% as 0..7 and 0.36. This is because, sampling of the tuples from every class caused one result tuple and all its dependent input tuples to get eliminated from the reduced provenance data. This reduced provenance data contains more amount of input tuples that contributed to the six tuples in the result with partial provenance and no provenance on one result tuple at all.

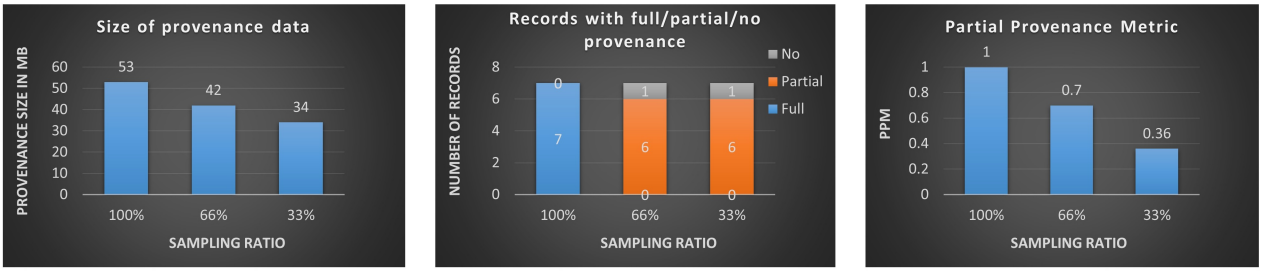
Scenario 6

In this scenario, the Spark program finds the average arrival delay per airport. We first execute the program and collect provenance for all tuples in the result. The number of tuples in the



(a) (b) (c)

Figure 5.30: Quantitative metrics for result of executing Spark program of Scenario 5 on data reduced by randomly eliminating 66% and 33% of the equivalence classes



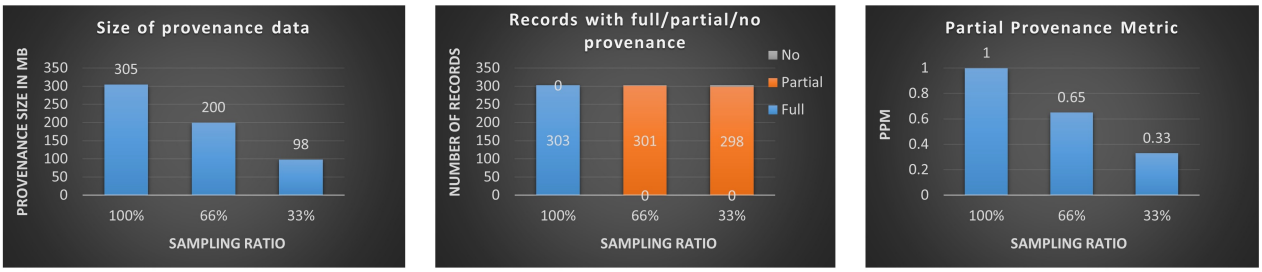
(a) (b) (c)

Figure 5.31: Quantitative metrics for result of executing Spark program of Scenario 5 on data reduced by eliminating records from equivalence classes

result set is 303. We then compute the quantitative metrics on this full provenance data. Then we apply the provenance data reduction techniques and collect partial provenance and compute these metrics on the partial provenance data and compare them with the full provenance data metrics.

Random sampling

We first apply random sampling to the input data and collect provenance for 66% and 33% of randomly selected input tuples. Figure 5.32 (a) shows the size of the provenance data collected for the different sampling ratios. The size of provenance data is linearly proportional to the sampling ratio. Figure 5.32 (b) shows the number of tuples in the result of the Spark program with full provenance, partial provenance and no provenance. For the reduced provenance of 66%, we notice there are 2 tuples with no provenance and 301 tuples with partial provenance. For the reduced provenance of 33%, the number of tuples in result with no provenance increases to 5 and the remaining 298 tuples have partial provenance. Figure 5.32 (c) shows that the partial provenance metric is almost equal to the sampling ratio.

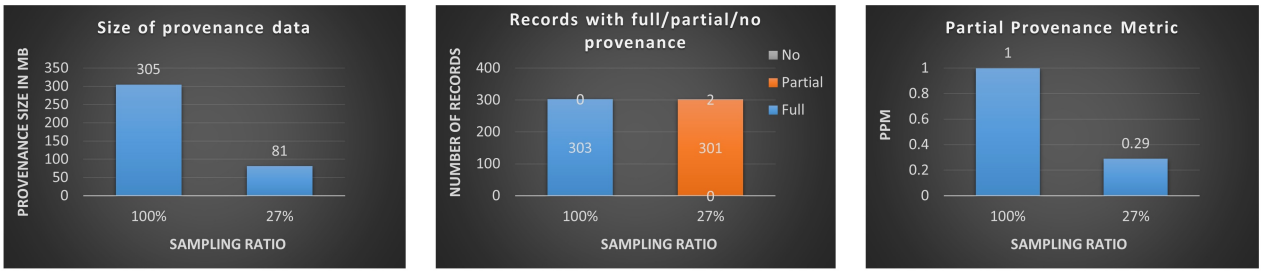


(a) (b) (c)

Figure 5.32: Quantitative metrics for result of executing Spark program of Scenario 6 on data reduced by random sampling

Histogram analysis

In histogram-based provenance data reduction, we compute provenance for only those tuples that have arrival delay in the range of most frequently occurring delay values. Such tuples account for 27% of the input data, i,e., Sampling ratio of 27%. Figure 5.33 (a) shows that the size of the provenance data collected for the 27% of the input tuples is 81MB. The size of the provenance data is linearly proportional to the sampling ratio or the number of input tuples for which we collect provenance. Figure 5.33 (b) shows that there are two tuples in the result which do not possess any provenance, and the rest of the 301 tuples possess partial provenance. Figure 5.33 (c) shows that the PPM is 0.29 for the tuples with provenance data sample of 27%. This is because the reduction retains most of the tuples related to the 301 tuples which possess partial provenance and completely eliminates all input tuples of the 2 tuples in result that possesses no provenance.

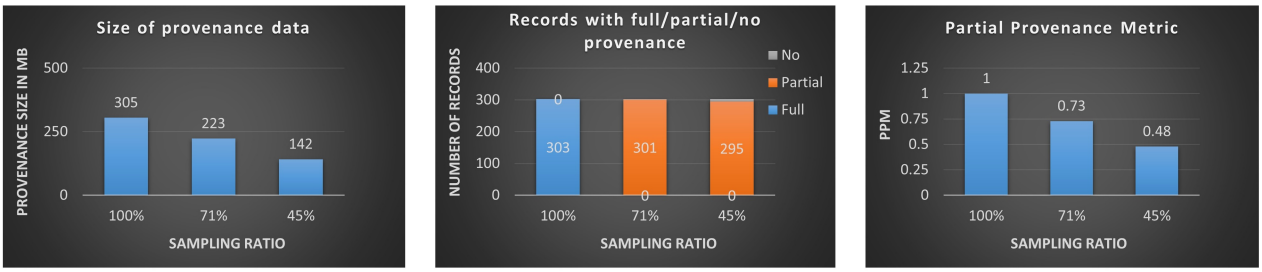


(a) (b) (c)

Figure 5.33: Quantitative metrics for result of executing Spark program of Scenario 6 on data reduced by histogram analysis

Clustering

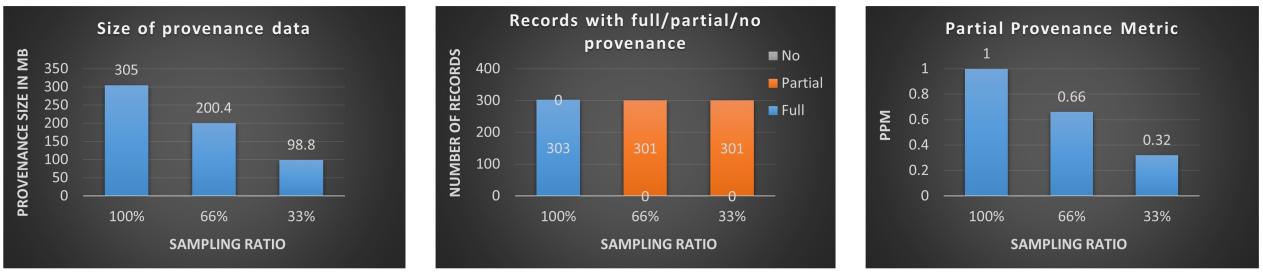
In clustering-based provenance data reduction, we can reduce data in two ways. Either by collecting provenance by only few clusters, or by collecting provenance for few tuples from every cluster. Figure 5.34 shows the results obtained by collecting provenance for tuples in few clusters. We randomly eliminate few clusters and collect provenance for the tuples present in rest of the clusters. We produce reduced input data of size 71% and 45%. Figure 5.34 (a) shows that the size of the provenance data obtained is linearly proportional to the sampling ratio. Figure 5.34 (b) shows that there are two tuples in the sample size of 71% that have no provenance. This number increases to 8 when we reduce the sample size to 45%. The corresponding PPM for these sample sizes is shown in Figure 5.34 (c). The PPM is 0.71 and 0.48 for the tuples, as clustering eliminates the input tuples that correspond to the tuple with no provenance completely and instead, collects provenance for most of the tuples that contribute to the tuples in result that possess partial provenance.



(a) (b) (c)

Figure 5.34: Quantitative metrics for result of executing Spark program of Scenario 3 on data reduced by randomly eliminating 66% and 33% of the clusters

In the second approach of clustering, we collect provenance for few tuples from each cluster. We collect provenance for 66% and 33% of tuples from each cluster. Figure 5.35 shows the metrics obtained for provenance data with 66% and 33% of input tuples from every cluster. Figure 5.35 (a) shows that the size of the provenance data is linearly proportional to the sampling ration. Figure 5.35 (b) shows that there are two tuples for sampling ratios of 66% and 33% of tuples without any provenance. Figure 5.35 (c) shows that the PPM is close to the sampling ratio.

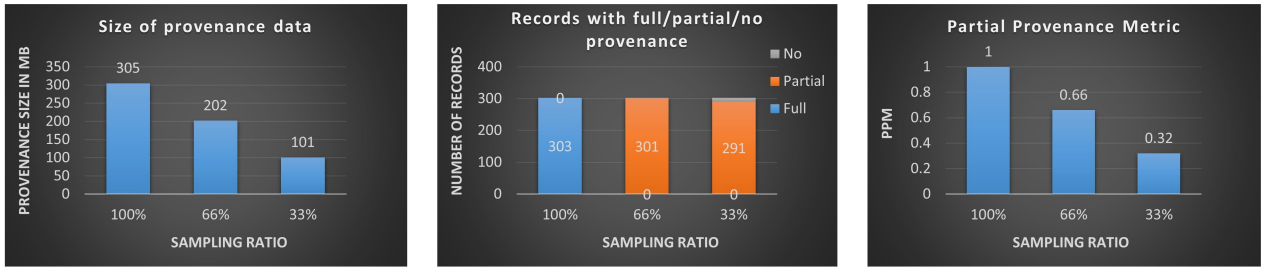


(a) (b) (c)

Figure 5.35: Quantitative metrics for result of executing Spark program of Scenario 6 on data reduced by eliminating records from clusters

Equivalence classes

In the equivalence class based approach, we perform reduction as in case of scenario 4 and 5. Figure 5.36 shows the metrics obtained by collecting provenance of all tuples that belong to few classes. The size of the provenance data is linearly proportional to the sampling ratio. We eliminate few classes and the remaining classes contain tuples that amount to 66% and 33% of the input tuples. Figure 5.36 (b) shows that in case of 66% provenance reduction, there are only two tuples without any provenance. However, when we reduce the provenance data to 33%, we see that there are 12 tuples that possess no provenance. Figure 5.36 (c) shows that the PPM is close to the sampling ratio. That is, the tuples in result that corresponds to 66% reduced provenance data contains 66% of full provenance.

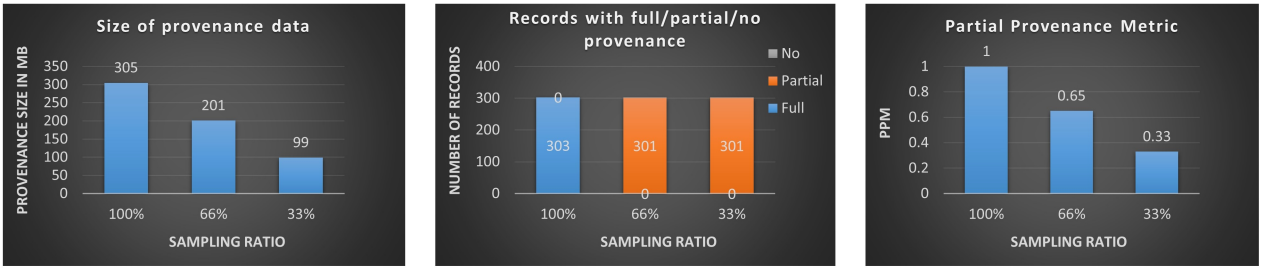


(a) (b) (c)

Figure 5.36: Quantitative metrics for result of executing Spark program of Scenario 6 on data reduced by randomly eliminating 66% and 33% of the equivalence classes

The metrics for the provenance data obtained by collecting provenance for few tuples from every class are similar to the metrics obtained by collecting provenance for all tuples from few classes, except for the number of records that possess no provenance. Figure 5.37 (c) shows the number of tuples in the result without provenance is equal to 2 for both sampling ratios of

66% and 33%. For the previous case, we see that this number is equal to 12. This is because we eliminate classes corresponding to attribute “year” in the previous case, so we are not collecting provenance for tuples that belong to a few years. The spark program calculates average delay per airport. So when we eliminate some years, few airports that were operating in those years were eliminated too. Hence we do not see such airports in the result. However, when we collect provenance for few tuples belonging to each year, there will be still be few tuples that contribute to such airports. Hence the number of tuples with no provenance decreases in the case of eliminating few tuples from each class.



(a) (b) (c)

Figure 5.37: Quantitative metrics for result of executing Spark program of Scenario 6 on data reduced by eliminating records from equivalence classes

5.5 Qualitative Analysis

In this section, we perform the qualitative analysis in order to find out whether we gain the same insights with reduced provenance obtained from different reduction techniques when compared to the insights we gain from full provenance. We specifically see how well the reduced provenance data answers the question(s) presented with the scenarios in Section 5.1 when compared to full provenance data. We first use full provenance to answer the question(s) proposed, then use partial provenance collected by applying random sampling, histograms, clustering and equivalence classes to answer the same question. We then argue on which data reduction technique works for the question and which does not and provide reasons.

5.5.1 Safecast Radiation dataset

In the following sections, we present the qualitative analysis for the scenarios of Safecast radiation dataset:

Scenario 1

In this scenario, the Spark program finds the 10 highest radiation levels per year between the years 2010 and 2016. The results of this program show that there is a sudden increase

in radiation levels in the year 2011 which made us pose a question - why is there a sudden increase in radiation level in the year 2011? We first use full provenance collected as part of execution of this program to answer this question. We find 318519 tuples in the full provenance corresponding to the year 2011 in the file of 1GB. The very high values of radiation occur after March 2011. When we filter the provenance data to retain only tuples after March 2011, the number of tuples reduces to 238889 tuples. Then we filter again on provenance data to retain only tuples whose radiation value is greater than 70000. We find 190672 such tuples. These tuples all correspond to the Fukushima region in Japan and were recorded when the nuclear disaster at Fukushima took place.

We then answer the provenance question using the reduced provenance data obtained by collecting provenance for only 66% and 33% randomly selected tuples from the input data. In this case, the number of tuples corresponding to the year 2011 are 206831 and 101463 respectively. Also, when we apply filter operation to retain tuples with radiation value more than 70000 as in the case of full provenance data, we see there are 125637 and 64728 tuples in case of reduced provenance data of 66% and 33% respectively. When we reduce the sampling ratio and collect provenance for only 1% of input tuples, we see close to 500 tuples that belong to year 2011 and have radiation value more than 70000. These 500 tuples are sufficient to answer the provenance question.

However, when we use histogram-based reduced provenance data, we see that for a sampling ratio of 36%, there are only 61363 tuples that belong to 2011. However there are no tuples that actually correspond to Fukushima disaster. This is because most tuples from 2011 do not fall into the range of most frequently occurring radiation value, as they correspond to the Fukushima nuclear disaster which does not occur frequently. Hence, histogram-based data reduction technique is not useful in answering this provenance question.

In case of clustering-based provenance data reduction, we either randomly eliminate complete clusters or few tuples from every cluster. We form clusters based on the latitude and longitude attributes of the tuples. We try to answer the question using reduced provenance obtained by eliminating complete clusters. In one of the runs, we randomly eliminate a cluster that corresponds to the Fukushima region. In this case, we are not able to answer the question because there is no provenance associated with the tuples that correspond to year 2011 and with radiation value more than 70000. However, when we randomly select 66% and 33% of tuples from every cluster, we can answer the question in the way similar to random sampling. We find these tuples corresponding to Fukushima disaster even in the case where sampling ratio is close to 2%. Hence, we can collect provenance for only 2% of the input tuples by performing clustering on it and still be able to answer the provenance question.

In the equivalence class based provenance data reduction, we form equivalence classes based on the attribute “year” of the tuple. We either collect provenance for all tuples of few classes or few tuples from all classes. In the former approach, we eliminate few randomly selected classes and collect provenance for the rest of them. So we collect provenance for tuples belonging to

only few years. When we eliminate classes randomly, we eliminate the tuples that belong to year 2011. In such a case, we are not able to answer the question as all the tuples from 2011 do not have provenance associated with them. When we eliminate randomly selected tuples from every cluster and collect provenance for few tuples, we have provenance for the tuples that correspond to the Fukushima nuclear disaster. We get the answer to this question in this approach similar to the random sampling approach, where even if the sampling ratio is 1%, we still have provenance for tuples that are sufficient to answer the provenance question.

Another question that is posed by examining the result of the scenario is - Why are all the ten highest radiation values belonging to 2016 equal to one another? When we use full provenance to answer this question, we first filter on provenance and select only records that correspond to 2016 and that have radiation value of more than 60000. These tuples all correspond to the same location and have same latitude and longitude values. We see that the number of such tuples is only 892. When we use reduced provenance data obtained by random sampling, we see that the number of these tuples reduces to 357 for sampling ratio of 66% and we do not see more than 20 of such tuples for sampling ratio of 33%. 20 tuples are too less to confidently claim that the radiation measurement must be an error from a faulty device even though those 20 tuples correspond to the same location. Hence, this question can be answered with random sampling only when the sampling ratio is close to 66%, that when we have provenance for close to or more than 66% of the input tuples.

The histogram-based provenance data reduction approach only collects provenance for the tuples that have most frequently occurring radiation values. The tuples that are interesting to us have radiation value more than 60000. Hence, we do not have provenance associated with such tuples as they do not hold one of the frequently occurring radiation values.

In case of clustering-based provenance data reduction approach, we form clusters based on location and eliminate them. We do not collect provenance for the tuples that correspond to the location with high values. Because there is no provenance associated with the tuples with the high radiation values of 2016, we cannot examine them and hence we are not able to answer the question. However, when we select tuples from every cluster, we observe that the question is answered in the same way as in random sampling.

In case of equivalence classes based provenance data reduction, we eliminate randomly the class that contains tuples belonging to year 2016. We do not possess provenance for the tuples belonging to year 2016 and hence we are not able to answer the question. In the other case, we collect provenance for few tuples from each class, i.e., each year. Hence, we see few tuples that belong to 2016 and have the value 64210. The answer to the question is obtained similar to the case of random sampling, where we find the answer only when the sampling ratio is close to or greater than 66%.

Scenario 2

In this scenario, the Spark program finds the average radiation level in U.S. We are interested particularly in a nuclear disaster that occurred in 2006 in Tennessee USA. However, when we examine the result of the program, we see that the average radiation level in the U.S in 2006 is not high as expected due to the nuclear disaster of Tennessee. So we pose a question - why is the average radiation level of U.S in 2006 not high? We try to answer this question using full provenance. But the number of tuples corresponding to U.S for the year 2006 are very few. Also these tuples are do not correspond to the location Tennessee. Hence they are not sufficient to provide any insight about the nuclear disaster. The same is the case with reduced provenance data, as it will contain less number of tuples than the full provenance and in this case, too, we do not obtain an answer. We apply the reduction techniques to the input data and collect reduced provenance. We use this reduced provenance to answer the provenance question. Using reduced provenance from all the reduction techniques, we gain the same insight as we get from full provenance. We find no tuples that correspond to Tennessee and hence we are not able to answer the provenance question.

Scenario 3

In this scenario, the spark program presents average radiation level per unit type. We then execute a program that counts the number of tuples that contain each of these types. We notice that the count program takes the longest on the “cpm” unit type, and hence we pose a question - why does the count program take the longest to run on “cpm” unit type? We use provenance to answer this question. We first use full provenance and filter on it to obtain tuples that correspond to each type. Then we count the number of tuples in each type and see that the number of tuples with “cpm” as unit is huge in comparison to other unit types. We see that 98.6% of the tuples in data have “cpm” as their unit type, hence any program executed on the tuples with “cpm” unit type will take longer than the tuples with other unit types. We then answer the question using provenance obtained by random sampling. Even when we reduce the sampling ratio to 1%, we still have tuples that correspond to “cpm” unit type, and their count is the largest in comparison to tuples with other unit types. Hence we are able to draw the same conclusion as in the case of full provenance that any program executed on “cpm” unit type will take the longest because the data contains most tuples that belong to the “cpm” type.

When we use the histogram based provenance data reduction, we collect provenance for the tuples whose radiation value falls in the range of most frequentlt occurring values. We find that most of the tuples belonging to this range have “cpm” as their unit type. There are very few tuples that have other unit types. Precisely, the number of tuples with “cpm” unit type that have provenance associated with them is 97%. When we reduce the size of the range of most frequently occurring radiation value, we still find tuples with “cpm” to be the ones that occur most often. In clustering based and equivalence class based approaches, we do not create clusters and classes based on attribute “unit type”. Hence we do not specifically eliminate tuples that belong to a certain unit type, thus excluding the possibility of the tuples of “cpm” unit type to be completely eliminated. Even in these cases, the answer to the provenance question is

obtained similar to the case of random sampling, where even with 1% of sampling ratio, we still can conclude that tuples with “cpm” as unit type are largest in number and hence any program executed on these tuples will take the longest when compared to tuples of other unit types.

5.5.2 U.S Domestic flights dataset

In the following sections, we present the qualitative analysis for the scenarios of U.S domestic flights dataset:

Scenario 4

In this scenario, the Spark program finds the average arrival delay per year for every carrier. Examining the result of the program, we notice that the delay is very high for the year 2006. We pose a question - why is the delay in 2006 the highest? We use provenance to answer this question. We first use full provenance and select the tuples that belong to the year 2006. We then search for the airlines that has the highest delay in 2006 and notice that “AA” and “UA” have the highest delays. When we examine the airport column associated with the tuples with these airlines and high delays, we find that it is “ORD”. The number of such tuples is 11163. When we use provenance data obtained by random sampling to answer this question, we see that as we reduce the sampling ratio, we lose provenance for these tuples that have “ORD” as the airport, “AA” or “UA” as airlines and belong to 2006. We see that we do not have such tuples when we reduce the sampling ratio to less than 8%.

In the histogram bases approach of data reduction, we collect provenance for the tuples that contain radiation value within the range of most frequently occurring values. However, we are interested in finding the tuples that caused maximum delay. Larger delay values do not occur frequently and hence we do not see any tuple that has large delay value and is still present in the range of most frequently occurring delay value. Hence, there is no provenance associated with the tuples that have large delay values in the case of histogram-based provenance data reduction.

In the clustering based and equivalence based approaches, we form clusters and classes based on the attribute “year”. In the first way of using these approaches, we eliminate few clusters or classes and collect provenance only for the tuples in the rest of the classes. This causes all the tuples belonging to a few years to not have provenance collected for them. When we used clustering and equivalence classes, we collected provenance for only a few clusters and classes which do not contain tuples belonging to year 2006. In this case, we are not able to answer the question using the reduced provenance obtained by using these approaches. In the other way of using these approaches, we randomly select few tuples from every cluster or class and collect provenance only for them. Thus we have few tuples, if not all, that correspond to high delay values and have “ORD” as the airport, “AA” or “UA” as airlines and belong to 2006. The answer to this question in this way is obtained similar to the random sampling approach, and we can achieve the answer only if the sampling ratio is higher than 8%.

Scenario 5

In this scenario, we have the Spark program that finds the average delay per day of the week. We see from the result of this program that the delay is high on Thursdays and Fridays, which poses a question - why is the delay high on Thursdays and Fridays? We first use full provenance collected for the results of this program to answer this question. We select only the tuples from Thursday and Friday from the provenance data. We notice that these tuples account for close to 40% of all the tuples in the data. The number of flights operating on these days is very high and hence the delay is high on these days. When we use the provenance data obtained from random sampling with sampling ratio of 66% and 33%, we see that the number of tuples that have day of the week as Thursday or Friday is always higher than the number of the tuples with other days. Even when we reduce the size of provenance data by decreasing the sampling ratio, the number of tuples with Thursday and Friday remains higher than the number of tuples with other days of the week. Thus we get an answer for our question even when we have small sampling ratios.

In the histogram based provenance data reduction, we are not able to get the answer to this question because we collect provenance only for most commonly occurring values of delay, and we are interested in high values of delay which does not occur often. In the clustering based and equivalence class based approaches, we get the answer to this question similar to the random sampling approach because we form the clusters and classes based on the year attribute and not the delay attribute. Hence we collect provenance for at least few tuples that have day of week as Thursday and Friday, which are still higher in number than the tuples that contain other days of the week.

Looking at the result of the Spark program of this scenario, we also notice that the delay is very low for the years 2001 and 2002. So we pose the question - why is the delay low for the years 2001 and 2002. We use provenance to find the answer to this question. We first use full provenance to answer the question, where we filter and inspect the provenance data related to the tuples belonging to years 2001 and 2002. We check and find that these flights correspond to dates after September 2001 until mid of 2002. So we conclude that the flights are the canceled flights after 9/11 attacks. When we use the provenance data obtained by random sampling, we see the tuples belonging to 2001 and 2002 even when we reduce the sampling ratio to 1%. Even though the number of such tuples in smaller samples will be low, we still get the answer to our question.

In the histogram based provenance data reduction, we collect provenance for tuples with most commonly occurring delay values. Few of the tuples that we are interested in have delay values that lie in the range of values which occur frequently. The number of tuples that fall in the range of most frequently occurring delay values amount to sampling ratio of 35%. In this case, we have 16372 cancelled flights between 2001 and 2002. Hence we are able to gain the same answer from both histogram based reduced provenance and full provenance.

In clustering and equivalence class based approaches, we create clusters and classes based on the attribute “year”, and collect provenance for all tuples that belong to few clusters or classes. In such a case, we randomly selected few clusters or classes that contain tuples belonging to a particular year and collected provenance only for them. We find that the years 2001 and 2002 are one of the few years whose tuples do not have provenance collected for them. Hence, we are not able to answer our provenance question. However, in the other way of clustering and equivalence class based approaches, we select few tuples from each cluster or class. Hence we have provenance for tuples from 2001 and 2002 which we are interested in. The answer to the question is obtained in the same way as in the case of random sampling. We can get answer to the question even when the sample size is 1% as there will always be few canceled flights with provenance in the time between September 2001 and 2002.

Scenario 6

In this scenario, the Spark program finds the average arrival delay per airport. We notice from the results that the values for average delay for few airports is in negative. So we pose a question - why are few values for average arrival delay in negative? We use provenance to answer this question. We backward trace these tuples and find that few tuples in the input have negative values for delay. These tuples with negative delays bias the average and in some cases make the average itself become negative. When we use provenance data obtained by randomly sampling input tuples and collecting provenance for only a few of them, we have provenance for these tuples that have negative delay values. However, when we reduce the sampling ratio to less than 25%, we do not have tuples with negative delay that has provenance associated with it. Hence, for sampling ratios more than 25%, we are able to answer the question on provenance data obtained by random sampling.

In histogram-based provenance data reduction, we form a histogram based on the most frequently occurring values of delay. The values we are interested in are the negative delay values, and they do not occur often. Hence, when we collect provenance for only frequently occurring delay values, we do not have provenance associated with the tuples that have negative values for delay. Hence, we cannot answer the question with provenance obtained by using histogram based data reduction.

In the case of clustering and equivalence classes approaches of provenance data reduction, the answer to the question is obtained in the same way as in random sampling. We do not form the clusters and the classes based on delay values and hence, we have few tuples with negative values for delay even after eliminating complete clusters and classes. Hence, in this case, we have few tuples with negative values until the sampling ratio is more than 25%.

Table 5.13 summarizes the qualitative analysis performed on all six scenarios. It describes how well a data reduction technique addresses the use case presented in the scenario. We have four broad categories of use cases of provenance that the six scenarios collectively address. We compare reduced provenance from random sampling, histogram, clustering and equivalence classes approaches on know if they provide the same insights as full provenance in the context of the given use case.

Use case of provenance	Scenario	Random sampling	Histogram	Clustering	Equivalence classes
Debugging	1	Provides same insights as full provenance data if sampling ratio is >66%	Does not provide same insights as full provenance data	Does not provide same insights as full provenance data if all tuples from a cluster are eliminated, or same as random sampling	Does not provide same insights as full provenance data if all tuples from a class are eliminated, or same as random sampling
	4	Provides same insights as full provenance data if sampling ratio is >8%	Does not provide same insights as full provenance data	Provides same insights as full provenance data only if >8% tuples are sampled from each cluster	Provides same insights as full provenance data only if >8% tuples are sampled from each class
	6	Provides same insights as full provenance data if sampling ratio is >25%	Does not provide same insights as full provenance data	Provides same insights as full provenance data only if >25% tuples are sampled from each cluster	Provides same insights as full provenance data only if >25% tuples are sampled from each class
Data exploration	1	Provides same insights as full provenance data if sampling ratio is >1%	Does not provide same insights as full provenance data	Does not provide same insights as full provenance data if the cluster containing tuples from a particular location are eliminated, or same as random sampling	Does not provide same insights as full provenance data if the class containing tuples from a particular year are eliminated, or same as random sampling
	2	Provides same insights as full provenance data	Provides same insights as full provenance data	Provides same insights as full provenance data	Provides same insights as full provenance data

	3	Provides same insights as full provenance data if sampling ratio is >1%	Provides same insights as full provenance data	Provides same insights as full provenance data	Provides same insights as full provenance data
	5	Provides same insights as full provenance data if sampling ratio is >1%	Does not provide same insights as full provenance data	Provides same insights as full provenance data if sampling ratio is >1%	Provides same insights as full provenance data if sampling ratio is >1%
Monitoring	3	Provides same insights as full provenance data if sampling ratio is >1%	Provides same insights as full provenance data	Provides same insights as full provenance data	Provides same insights as full provenance data
	5	Provides same insights as full provenance data if sampling ratio is >1%	Provides same insights as full provenance data	Does not provide same insights as full provenance data if the cluster containing tuples from a particular year are eliminated, or same as random sampling	Does not provide same insights as full provenance data if the class containing tuples from a particular year are eliminated, or same as random sampling
Data quality	2	Provides same insights as full provenance data	Provides same insights as full provenance data	Provides same insights as full provenance data	Provides same insights as full provenance data
	6	Provides same insights as full provenance data if sampling ratio is >25%	Does not provide same insights as full provenance data	Provides same insights as full provenance data only if >25% tuples are sampled from each cluster	Provides same insights as full provenance data only if >25% tuples are sampled from each class

Table 5.13: Summary on qualitative analysis

6 Conclusion and Future Work

In this thesis, we present nine broad categories of use-cases and applications of provenance. We describe how provenance is used in these applications. Insights obtained from such analysis helps to find suitable provenance data reduction techniques for these applications. One such insight is that many applications like data security require fine grained provenance for every tuple in the input and hence they provide less opportunity for data reduction as provenance is necessary for all of the input. However, there are data-agnostic applications of provenance like data exploration, monitoring etc., where reduced provenance in which collecting only a subset of the fine-grained provenance is sufficient. Based on such insights, we propose provenance data reduction techniques that are useful in data-agnostic applications of provenance.

We introduce the techniques - deduplication, outlier detection, stream summary, count-min sketch and locality sensitive hashing, and have a closer look at these four techniques - sampling, histogram, clustering, equivalence classes. For the evaluation, we introduce two datasets and six different scenarios and test the four techniques on top of Spark using Titian. The scenario description also presents the applications and use-cases the scenario is useful in. There are four broad categories of applications these scenarios cover - data exploration, monitoring, data quality and debugging. We perform a quantitative and qualitative analysis of the reduction techniques in each scenario. In the quantitative analysis, we compare reduced provenance data obtained by different techniques based on 3 metrics we introduce. These metrics quantify the size of reduced provenance data and the amount of provenance it possesses. In qualitative analysis, we compare different data reduction techniques based on how well they address the use case of provenance.

We notice that the size of the provenance data collected and the amount of provenance obtained after reduction is proportional to the sampling ratio in all scenarios and for all data reduction techniques. The provenance data obtained by applying random sampling addresses the scenarios that are useful in data exploration better than other techniques in case the sampling ratio is more than 5%. Clustering and equivalence class based approaches do not work well in the field of data exploration if the attribute based on which the clusters and classes are formed is also the attribute in data the application of provenance is interested in. This is because we perform reduction based on the attributes the clusters and classes are created from, and there is a possibility that we do not collect provenance for the tuples that hold a certain attribute the application is interested in. All the techniques work equally well for monitoring based applications like latency profiling. The histogram approach does not work well when provenance data is used for data exploration, debugging and data quality when compared to other reduction techniques. Because the data values these applications are interested in do not fall into the most frequently occurring data values, based on whom we perform reduction. Clustering and equivalence class techniques work best for debugging when the clusters and classes formed are based on the attribute the

debugging application is interested in.

However, to conclude on what technique works best always for a particular application of provenance, we have to perform extensive evaluation and analysis which can be extended as part of future work. Adding more scenarios that would be useful in other applications of provenance and performing thorough analysis on them would help in this direction. We have formally described many other techniques which have not been evaluated as part of this thesis. We can evaluate them and compare their performance with the current techniques and see which of them would address the application of provenance in the best way. We have collected provenance using Titian and performed the evaluation on top of Spark. We can instead perform evaluation within Spark and embed the reduction techniques within Titian such that it collects less provenance data based on the technique chosen. The current version of Titian limits provenance collection to a fixed subset of Spark operators. Hence, it restricts us in building complex and sophisticated programs to realize the scenarios. We can perform the evaluation with more complex programs that would support more comprehensive analysis of each of the techniques.

Bibliography

[AA03] A. Aron, E. N. Aron. *Statistics for psychology* . Prentice Hall/Pearson Education, 2003 (cit. on p. 51).

[ACT06] B. Alexe, L. Chiticariu, W.-C. Tan. “Spider: a schema mapping debugger.” In: *Proceedings of VLDB Endowment (PVLDB)*. 2006 (cit. on p. 18).

[Ain+14] E. Ainy, S. B. Davidson, D. Deutch, T. Milo. “Approximated Provenance for Complex Applications.” In: *Theory and Practice of Provenance (TaPP)*. 2014 (cit. on pp. 12, 25–27, 33, 34).

[All97] M. P. Allen. “Populations, samples, and sampling distributions.” In: *Understanding Regression Analysis* (1997), pp. 51–55 (cit. on p. 48).

[Alp+13] P. Alper, K. Belhajjame, C. A. Goble, P. Karagoz. “Enhancing and abstracting scientific workflow provenance for data publishing.” In: *Proceedings of the Joint EDBT/ICDT Workshops*. 2013 (cit. on pp. 12, 30, 33–35).

[Ams+11] Y. Amsterdamer, S. B. Davidson, D. Deutch, T. Milo, J. Stoyanovich, V. Tannen. “Putting lipstick on pig: Enabling database-style workflow provenance.” In: *Proceedings of the VLDB Endowment (PVLDB)* (2011) (cit. on pp. 11, 16, 18, 21, 22, 25, 40).

[BKWC01] P. Buneman, S. Khanna, T. Wang-Chiew. “Why and where: A characterization of data provenance.” In: *International conference on database theory (ICDT)*. 2001 (cit. on p. 17).

[Bun+95] P. Buneman, S. Naqvi, V. Tannen, L. Wong. “Principles of programming with complex objects and collection types.” In: *Theoretical Computer Science* (1995) (cit. on p. 22).

[CCBD06] S. Cohen, S. Cohen-Boulakia, S. Davidson. “Towards a model of provenance and user views in scientific workflows.” In: *International Workshop on Data Integration in the Life Sciences (DILS)*. 2006 (cit. on p. 17).

[CM05] G. Cormode, S Muthukrishnan. “An improved data stream summary: the count-min sketch and its applications.” In: *Journal of Algorithms* (2005) (cit. on p. 62).

[CSH07] B. J. Corcoran, N. Swamy, M. Hicks. “Combining provenance and security policies in a web-based document management system.” In: *On-line Proceedings of the Workshop on Principles of Provenance (PrOPr)*. 2007 (cit. on p. 40).

[CWW00] Y. Cui, J. Widom, J. L. Wiener. “Tracing the lineage of view data in a warehousing environment.” In: *ACM Transactions on Database Systems (TODS)* (2000) (cit. on p. 18).

[Che+16] W. Chen, G. Yin, G. Zhao, Q. Han, W. Jing, G. Sun, Z. Lu. *Big Data Technologies and Applications*. Springer, 2016 (cit. on p. 20).

[Cho+16] Z. Chothia, J. Liagouris, F. McSherry, T. Roscoe. “Explaining Outputs in Modern Data Analytics.” In: *Proceedings of the VLDB Endowment (PVLDB)* (2016) (cit. on pp. 18, 22–25).

[DG08] J. Dean, S. Ghemawat. “MapReduce: simplified data processing on large clusters.” In: *Communications of the ACM (CACM)* (2008) (cit. on p. 19).

[DMT14] D. Deutch, Y. Moskovitch, V. Tannen. “A provenance framework for data-dependent process analysis.” In: *Proceedings of the VLDB Endowment (PVLDB)* (2014) (cit. on pp. 12, 27–29, 33, 34).

[De12] S. De. “Newt: an architecture for lineage-based replay and debugging in DISC systems.” In: *Symposium on Cloud Computing (SoCC)* (2012) (cit. on p. 20).

[Fos+03] I. T. Foster, J.-S. Vöckler, M. Wilde, Y. Zhao. “The Virtual Data Grid: A New Model and Architecture for Data-Intensive Collaboration.” In: *Conference on Innovative Data Systems Research (CIDR)*. 2003 (cit. on p. 41).

[GKT07] T. J. Green, G. Karvounarakis, V. Tannen. “Provenance semirings.” In: *ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*. 2007 (cit. on p. 22).

[Gal+01] H. Galhardas, D. Florescu, D. Shasha, E. Simon, C.-A. Saita. “Improving Data Cleaning Quality Using a Data Lineage Facility.” In: *Design and Management of Data Warehouses (DMDW)*. 2001 (cit. on p. 39).

[Gla+13] B. Glavic, K. Sheykh Esmaili, P. M. Fischer, N. Tatbul. “Ariadne: Managing fine-grained provenance on data streams.” In: *ACM International Conference on Distributed event-based systems (DEBS)*. 2013 (cit. on pp. 12, 31–34).

[Gro+05] P. Groth, S. Miles, W. Fang, S. C. Wong, K.-P. Zauner, L. Moreau. “Recording and using provenance in a protein compressibility experiment.” In: *IEEE International Symposium on High Performance Distributed Computing (HPDC)*. 2005 (cit. on p. 37).

[HDBL17] M Herschel, R Diestelkämper, Ben-Lahmar. “A survey on provenance - What for? What form? What from?” In: *yet to be published* (2017) (cit. on p. 35).

[HH16] M. Herschel, M. Hlawatsch. “Provenance: On and Behind the Screens.” In: *International Conference on Management of Data (ICMD)*. 2016 (cit. on pp. 15, 16).

[Haw80] D. M. Hawkins. *Identification of outliers*. Vol. 11. Springer, 1980 (cit. on pp. 60, 61).

[Her15] M. Herschel. “A hybrid approach to answering why-not questions on relational query results.” In: *Journal of Data and Information Quality (JDIQ)* (2015) (cit. on p. 15).

[IPW11] R. Ikeda, H. Park, J. Widom. “Provenance for generalized map and reduce workflows.” In: *Conference on Innovative Data Systems Research (CIDR)* (2011) (cit. on pp. 11, 16, 18, 19, 22, 25).

[Int+15] M. Interlandi, K. Shah, S. D. Tetali, M. A. Gulzar, S. Yoo, M. Kim, T. Millstein, T. Condie. “Titian: Data provenance support in spark.” In: *Proceedings of the VLDB Endowment (PVLDB)* (2015) (cit. on pp. 11, 12, 16, 18, 20–22, 24, 25, 45, 65, 76).

[JMF99] A. K. Jain, M. N. Murty, P. J. Flynn. “Data clustering: a review.” In: *ACM computing surveys (CSUR)* (1999) (cit. on pp. 54, 55).

[JO04] H. Jagadish, F. Olken. “Database management for life sciences research.” In: *ACM SIGMOD Record* 33.2 (2004) (cit. on pp. 39, 42).

[LDY13] D. Logothetis, S. De, K. Yocum. “Scalable lineage capture for debugging disc analytics.” In: *Symposium on Cloud Computing (SoCC)*. 2013 (cit. on pp. 11, 18, 22, 24, 25).

[LRU14] J. Leskovec, A. Rajaraman, J. D. Ullman. *Mining of massive datasets*. Cambridge University Press, 2014 (cit. on p. 63).

[LW15] S. Li, X. Wang. “Study on dangerous signal mining of dangerous goods transport vehicles.” In: *International Journal of Control and Automation (IJCA)* (2015) (cit. on p. 54).

[MAEA05] A. Metwally, D. Agrawal, A. El Abbadi. “Efficient computation of frequent and top-k elements in data streams.” In: *International conference on database theory (ICDT)*. 2005 (cit. on p. 61).

[MB12] D. T. Meyer, W. J. Bolosky. “A study of practical deduplication.” In: *Transactions on Storage (TOS)* (2012) (cit. on p. 60).

[MP12] Z. Manna, A. Pnueli. *The temporal logic of reactive and concurrent systems: Specification*. Springer Science & Business Media, 2012 (cit. on p. 28).

[Mac+67] J. MacQueen et al. “Some methods for classification and analysis of multivariate observations.” In: *Berkeley symposium on mathematical statistics and probability*. 1967 (cit. on p. 55).

[OR11] C. Olston, B. Reed. “Inspector gadget: A framework for custom monitoring and debugging of distributed dataflows.” In: *International Conference on Management of data (ICMD)*. 2011 (cit. on pp. 36–38).

[Ols+08] C. Olston, B. Reed, U. Srivastava, R. Kumar, A. Tomkins. “Pig latin: a not-so-foreign language for data processing.” In: *International Conference on Management of Data (ICMD)*. 2008 (cit. on pp. 11, 18, 21).

[PIW11] H. Park, R. Ikeda, J. Widom. “Ramp: A system for capturing and tracing provenance in mapreduce workflows.” In: (2011) (cit. on p. 19).

[PMS14] S. Prasad, S. Mathur, S. Singh. “Border security up gradation using data mining.” In: *International Journal of Soft Computing and Engineering (JSCSE)* (2014) (cit. on p. 54).

[SEA14] D. Smith, M. Eggen, R. S. Andre. *A transition to advanced mathematics*. Nelson Education, 2014 (cit. on p. 57).

[SPG05] Y. L. Simmhan, B. Plale, D. Gannon. “A survey of data provenance in e-science.” In: *ACM Sigmod Record* (2005) (cit. on pp. 11, 17, 18).

[Shv+10] K. Shvachko, H. Kuang, S. Radia, R. Chansler. “The hadoop distributed file system.” In: *IEEE on Mass storage systems and technologies (MSST)*. 2010 (cit. on pp. 11, 18).

[TKH13] Y. S. Tan, R. K. Ko, G. Holmes. “Security and data accountability in distributed systems: A provenance survey.” In: *IEEE International Conference on High Performance Computing and Communications (HPCC) and IEEE International Conference on Embedded and Ubiquitous Computing (EUC)*. 2013 (cit. on p. 41).

[Al-14] Al-Kashi. *Background Statistics*. 2014. URL: http://www.ar-php.org/stats/al-kashi/doc/Al-Kashi_Background_Statistics.pdf (cit. on p. 51).

[Ily12] Ilya Katsov. *PROBABILISTIC DATA STRUCTURES FOR WEB ANALYTICS AND DATA MINING*. 2012. URL: https://highlyscalable.wordpress.com/2012/05/01/probabilistic-structures-web-analytics-data-mining/ (cit. on p. 62).

[Mac11] Maciej Pacula. *k-means clustering example*. 2011. URL: http://blog.mpacula.com/?s=k+means (cit. on p. 55).

[Mar16] Margaret Rouse. *Data sampling*. 2016. URL: http://searchbusinessanalytics.techtarget.com/definition/data-sampling (cit. on pp. 45, 46).

[Pas12] Passy. *World of Mathematics*. 2012. URL: http://passyworldofmathematics.com/grouped-data-histograms/ (cit. on p. 52).

[Saf16] Safecast. *Data on radiation*. 2016. URL: http://blog.safecast.org/data/ (cit. on p. 65).

[The16] The Apache Software Foundation. *Apache Spark - Lightning-Fast Cluster Computing*. 2016. URL: http://www.spark.apache.org (cit. on pp. 11, 15, 18, 20, 65).

[Vin14] Vindika Lokunarangodage. *Food Sampling and Analysis*. 2014. URL: http://iso22000resourcecenter.blogspot.sg/2014/06/food-sampling-analysis-ii.html (cit. on p. 49).

All links were last followed on January 30, 2017.

Declaration

I hereby declare that the work presented in this thesis is entirely my own and that I did not use any other sources and references than the listed ones. I have marked all direct or indirect statements from other sources contained therein as quotations. Neither this work nor significant parts of it were part of another examination procedure. I have not published this work in whole or in part before. The electronic copy is consistent with all submitted copies.

---

place, date, signature