**Noname manuscript No.**
(will be inserted by the editor)

# A survey on provenance

## What for? What form? What from?

**Melanie Herschel · Ralf Diestelkämper · Houssem Ben Lahmar**

Received: date / Accepted: date

**Abstract** Provenance refers to any information describing the production process of an end product, which can be anything from a piece of digital data to a physical object. While this survey focuses on the former type of end product, this definition still leaves room for many different interpretations of and approaches to provenance. These are typically motivated by different application domains for provenance (e.g., accountability, reproducibility, process debugging) and varying technical requirements such as runtime, scalability, or privacy. As a result, we observe a wide variety of provenance types and provenance-generating methods.

This survey provides an overview of the research field of provenance, focusing on what provenance is used for (*what for?*), what types of provenance have been defined and captured for the different applications (*what form?*), and which resources and system requirements impact the choice of deploying a particular provenance solution (*what from?*). For each of these three key questions, we provide a classification and review the state of the art for each class. We conclude with a summary and possible future research challenges.

## 1 Provenance: data about a production process

Provenance generally refers to any information that describes the production process of an end product, which can be anything from a piece of data to a physical object. Thus, provenance information encompasses meta-data about entities, data, processes, activities, and persons involved in the production process [89]. Essentially, provenance can be seen as meta-data that, instead of describing data, describes a production process. The collection (also referred to as *capture*) and processing of provenance is important in various settings, e.g., to assess quality, to ensure reproducibility, or to reinforce trust in the end product. As a first glimpse on the large variety of possible provenance applications, consider the following three sample use cases.

### 1.1 Provenance use cases *(What for?)*

**Supply chains.** In 2013, the European food market was hit by a major scandal: several processed food products supposedly containing beef were discovered to contain horse meat instead. As a consequence, labelling the origin of (ingredients in processed) food and documenting its whole processing and supply chain were debated by food vendors, national governments, and the European Commission [53]. Independently of the individual measures taken, consumers experienced a (at least temporary) general awareness and increased need for information on the provenance of food to better assess food's quality and to regain trust in food products. Similar stories regarding how provenance about product origin and supply chain can be told for various other sectors [146].

**Scientific experiments.** The ATLAS experiment is one major experiment at the Large Hadron Collider at CERN. The first run of data acquisition has led to the accumulation of around 100 PB of data (both raw and processed). Experiments of this scale are not easily repeatable, meaning that ensuring the reproducibility and accessibility of the scientific results is essential. Therefore, the ATLAS collaboration has set up a strategy for preserving both the data and the procedures used to analyze them, relying on provenance [54].

**Complex data processing.** When implementing applications involving data processing, it is typically difficult to formulate the correct queries or data manipulations from scratch at first try. Reasons for instance lie in a poor documentation of used sources, cryptic values, unclear data qual-

M. Herschel, R. Diestelkämper, and H. Ben-Lahmar,
University of Stuttgart, Universitätsstr. 38, 70569 Stuttgart, Germany
E-mail: {firstname.lastname}@ipvs.uni-stuttgart.de

ity, faulty code, etc. This entails incremental development of complex data processing pipelines: developers write the necessary code, test it, analyze the result, refine the code (or clean the data) as the result does not (yet) match their expectation, test again and so on until the desired and expected result is obtained. The collection and analysis of provenance in this context can lead to faster and higher-quality analysis, debugging, and refinement of data processing [30, 96].

From the above three examples, we observe that the end product for which provenance is collected can be anything (food, a scientific result, or derived data). This survey focuses on methods and systems providing provenance for derived data that result from possibly complex data processing pipelines. To the best of our knowledge, this is also the focus of existing systems that support provenance.

The above examples further illustrate some of the different applications of provenance, e.g., assessing quality, preserving processes for repeatability, or supporting query development to improve process quality. This survey provides a classification of the various applications of provenance that in general serve to understand, reproduce, and improve production processes and their end products.

## 1.2 Types of provenance *(What form?)*

Within the scope of considered provenance use cases, i.e., provenance of derived data serving various applications, we observe that different types of provenance can be collected. This survey presents a classification of these provenance types. Based on [98], our classification includes four main provenance types, namely *provenance meta-data*, *information systems provenance*, *workflow provenance* and *data provenance*. As illustrated in Fig. 1, these roughly form a hierarchy where provenance meta-data is the widest, i.e., most general type, whereas data provenance has the most specific domain. That is, as we move up the hierarchy, additional restrictions on the provenance type apply, reducing some degrees of freedom at the benefit of being able to leverage the narrower scope to collect provenance. More specifically, these restrictions limit the set of supported processes or provenance models (see left side of Fig. 1) and directly relate to the level of instrumentation available to collect provenance of a given type (see right side of figure). Indeed, at the low level of the spectrum, no instrumentation may be available in general (and thus, provenance collection reduces to some ad-hoc meta-data collected about an arbitrary production process). Opposed to that, data provenance relies on a highly instrumented environment, exploiting clear semantics of individual processing steps and relationships between them available due to the restriction to structured data models and associated declarative query languages.

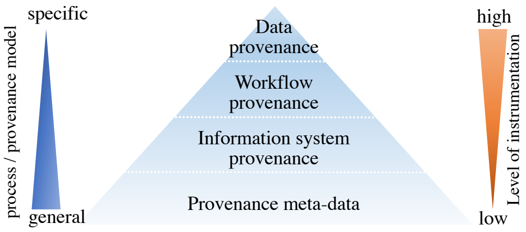

Fig. 1: Provenance hierarchy

We further sharpen the distinction between the different provenance types below. However, we point out that the transition from one type to another is a gradient rather than a sharp edge. The detailed discussion of the survey will then focus on the two top-levels of the hierarchy, namely data provenance and workflow provenance.

**Provenance meta-data.** Provenance meta-data is the most general type of provenance that encompasses any possible meta-data about a production process. It provides users with the widest degree of freedom to model, store, or access the provenance of any type of end product or production process and allows to classify proprietary solutions for provenance management (e.g., Informatica Big Data Management, IBM InfoSphere Information Governance Catalog, SAS Lineage) where internals are not disclosed. Note that we distinguish provenance meta-data from other meta-data based on their intended applications. Indeed, whereas general meta-data aims at assigning meaning to data, provenance information is descriptive of the data derivation process [64].

Being a placeholder for any provenance, no restrictions nor assumptions apply on the underlying process being described by provenance, the data model of provenance information, etc. Consequently, we cannot instrument any predefined properties or assumptions for the purpose of provenance management. So in summary, provenance meta-data is defined as meta-data describing an arbitrary production process using an arbitrary data model and model of computation.

**Information system provenance.** Moving up one level in the hierarchy, we restrict the type of production process to processes producing digital data within information systems (in a very wide sense, ranging from the Web over networks to Big Data processing pipelines). Information system provenance is meta-data about processes within an information system that are involved in the dissemination of information (e.g., storage / retrieval [50, 142], communication [68, 123, 141, 190, 192], or distribution [113, 140, 191] of information). While the internals of each process are generally unknown, provenance collection may exploit the input, the output, and parameters of a process.

In summary, we define information system provenance as meta-data collected for an information-disseminating process that can be computed based on the input, the output, and the parameters of the process.

**Workflow provenance.** Workflow provenance, surveyed for instance in [62, 74], specializes the previously described information system provenance by further restricting the type of production processes to so called workflows. In this survey, we view a workflow as a directed graph where nodes represent arbitrary functions or modules in general with some input, output, and parameters and where edges model a predefined dataflow or control flow between these modules. Given this more detailed process model, the degree of instrumentation for provenance collection increases as systems supporting workflow provenance leverage all information of the workflow graph. As we will see, this rich information (as compared to previously discussed provenance types) allows for different forms and granularities of provenance in different application domains where characteristics of the workflow graph vary.

**Data provenance.** Data provenance (surveys include [46]) allows to track the processing of individual data items (e.g., tuples) at the "highest resolution", i.e., the provenance itself is at the level of individual data items (and the operations they undergo). Collecting data provenance typically applies on structured data models and declarative query languages with clearly defined semantics of individual operators. This is necessary to push the degree of instrumentation to its highest level, where, on top of the instrumented information for workflow provenance, we now also benefit from having processes with clear semantics (based on an algebra, calculus, or other formalism). This high degree of instrumentation allows to obtain data provenance of individual data items.

Due to space constraints, we only briefly summarize essential work on data provenance recently surveyed in [46] to the benefit of a more detailed novel survey on provenance research focusing on explaining why some data are not part of a query result, even though they were expected results.

### 1.3 System requirements *(What from?)*

Collecting, querying, or analyzing provenance to achieve the intended application generally adds additional load on any data processing system. One reason for why research on provenance has not yet been widely adopted by practical applications may be that the system requirements entailed by proposed algorithms are not transparent enough or do not fit the expectations of system administrators. Intuitively, one question to be answered is *"From what requirements or resources may my choice of provenance technique depend?"*

In this survey, we consider various system requirements, including (but not limited to) runtime, scalability, security, interoperability, system decoupling, or fault tolerance. In light of these requirements, we analyze which of these have been considered or even optimized in the research literature.

### 1.4 Contributions and article structure

As illustrated by our use cases, provenance is an important topic that has high practical relevance in various areas of data management and beyond. Its timeliness and importance are tightly coupled with the increasing number and complexity of data-driven processes entailed by recent trends (Big Data analytics, Internet of Things).

Compared to previous surveys on provenance that, except for [136], focus on a particular type of provenance, this survey summarizes and characterizes work across provenance types, providing a broad view of the field. This perspective allows us to more precisely set terms that have until now been used in an ad-hoc and often inconsistent way in the provenance community. Finally, this survey puts a particular focus on recent provenance research and thus complements earlier surveys that predate recent trends such as distributed big data processing or process debugging. Overall, we provide an extensive and up to date classification of provenance research. Note that in analogy to our earlier comment on provenance types, while our discussion will generally strictly follow the proposed classes, solutions may fall in several classes and transitions and distinctions are more of a gradient. Based on our detailed study, we derive several research opportunities in the area of provenance.

Sec. 2 provides a classification of applications of provenance. Sec. 3 and 4 cover research on data provenance and workflow provenance in detail. Which factors of the system environment can or should be considered when designing or extending a system that leverages provenance is at the heart of Sec. 5. We conclude this survey by a summary and an outlook of provenance research challenges in Sec. 6.

## 2 Applications of provenance

As illustrated in Sec. 1.1, provenance may serve various applications. This section provides a classification of applications of provenance found in the literature, which is summarized in Fig. 2. This classification is built on the general idea of first understanding, then reproducing and validating a process, before it can eventually be improved. This is reflected by the first layer of the classification that distinguishes between *understandability*, *reproducibility*, and *quality*. The second layer further distinguishes applications of provenance based on their "target audience", i.e., the typical users capturing and using the provenance. While the classes of our taxonomy are shown in blue in Fig. 2, producers and consumers of provenance are labelled below the yellow arrows. We identify three types of involved parties:

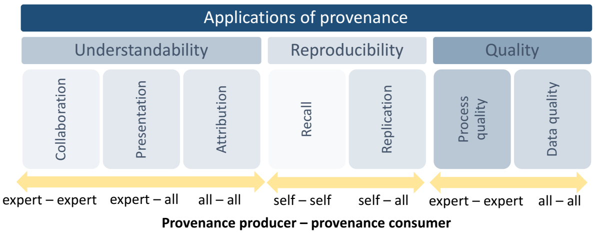

Fig. 2: Classification of applications of provenance

*expert* designates the entities that implement or administer the processing for which provenance is collected, *all* encompasses both these experts as well as end users of the end product (not necessarily involved or knowledgeable in how it was obtained), and *self* restricts the producer and consumer to be the same expert. The distinction of user types may allow the development of applications tailored and optimized to different configurations of provenance producers and consumers in the future.

While this section focuses on describing applications for which provenance solutions have been developed, note that solutions proposed for one application may equally serve other applications as well. This is due to the fact that the same general challenges underlie these applications. For instance, the reconstruction of intermediate processing states supports both an expert in recalling why different processing steps were necessary to obtain a result and equally allows scientists to replicate each other's result step by step. Knowledge about intermediate processing states also facilitates the further improvement of a process. Another overarching challenge is the attribution of data to sources or processes. Clearly, this helps in attributing changes and new versions of a process to collaborating developers and allows a more transparent and verifiable presentation of derived facts. This may ultimately impact the trust in data, trust being one important data quality dimension. As third example, communicating changes in a collaborative setting via provenance is relevant for collaboration and replication of the process by all collaborators, and bears important information for process and product quality assessment and improvement.

Despite these common challenges, we observe that within the literature, a solution is often focused on addressing these challenges within the scope of a particular application. This may be caused by the different environments, types of processes, or properties of the solutions themselves. Therefore, and further being inspired by previous classifications of provenance applications and applications [83, 87, 162, 166], our discussion is structured around particular applications. Our classification is the first that is domain-independent and that takes into account the latest developments in provenance research of the last decade. Indeed, the classifications of application of provenance from the mid 2000's [83, 166] focus on scientific and business domains while application classes in [87, 162] are guided by possible uses of provenance for Web data and in visualization tools, respectively. Finally, our classification considers the entities capturing and consuming provenance.

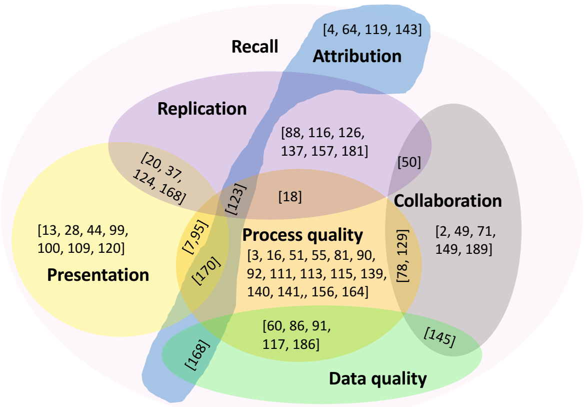

Fig. 3: Survey of applications of provenance in the literature

We now discuss the different classes of our application classification in more detail. Fig. 3 summarizes works referenced for each application as examples for solutions for a given application. We neither claim that the list is exhaustive nor that a system targeted or tested for one application may not equally work for another (illustrated by several examples in intersections). Indeed, as we have seen, solutions covering a challenge shared among applications may span multiple classes.

## 2.1 Understandability

*Understandability* emphasizes on explaining results or processes to make these transparent to some audience. Research on understandability includes to identify what information to convey and how. In this context, provenance conveys information on how an end product was obtained to expert users or a general audience (the provenance consumers). In this survey, this application class unifies applications related to understandability previously identified in [162, 166].

**Collaboration.** To collaborate among multiple users working on a same project, understanding the actions of collaborators is key. As described in [162], provenance may convey relevant information to allow each group member to better understand the actions of others. Clearly, this application of provenance is centered around expert groups within which each member produces and consumes provenance.

We further distinguish between *synchronous* and *asynchronous* collaboration. Essentially, synchronous collaboration requires the communication of information, including provenance, among group members *in real-time*, whereas asynchronous collaboration does not. Systems leveraging provenance in both settings exist. For instance, the infrastructure presented in [71] captures, during collaborative design, provenance about the evolution of a workflow based on tracking changes made by all users in real-time. Unlike version control systems like GIT [1] and Mercurial [2], [71] does not merge changes automatically. Instead, it renders a workflow evolution provenance containing branches that present users' changes. Asynchronous collaboration typically involves exporting provenance and making it accessible and query-able to reviewers or collaborators [49, 50, 149, 189].

As the size of provenance can become large, it may easily overwhelm users, defeating the purpose of understandability. As counter-measure, summarization and aggregation of provenance have been explored (e.g., [2, 78]).

In a collaborative setting, preserving privacy and confidentiality of information is essential and also spans information contained in provenance traces. Approaches considering these issues include [129, 145].

**Presentation.** The way information is presented and displayed significantly impacts on its understandability, requiring research on appropriate visualization of and interaction with the data. In this context, enriching data with information on how they were obtained may potentially improve on their understandability. This means using provenance for *presentation* [162] (an application also called information [166]). This application of provenance is tightly linked to the use of visualizations that allow to easily display, navigate, and explore provenance.

Node link graphs have been widely adopted to visualize provenance [13, 20, 37, 44, 109, 169]. A drawback of this technique is its scalability to large graphs, as it leads to cluttered visualizations for large provenance data sets. The use of Sankey diagram addresses this scalability issue [100] by grouping nodes of the node-link provenance graphs and highlighting the magnitude of data flows exchanged between activities. Further explored visualizations include radial form [28] and Lamport diagrams [7]. Whereas the former is suited to show relationships between and navigate through different resolutions of provenance, the latter is used to present provenance tracked in distributed systems with multiple actors and at different points in time.

Hlawatsch et al. [99] combine many visualization techniques in order to ensure scalability of the presentation to large provenance traces collected for workflows. The visualization emphasizes the logical time order of provenance, employs summarization techniques and visualizes branches to highlight the workflow evolution and dependencies.

We further differentiate works for provenance-based presentation based on their interaction capabilities. Systems either limit to static visualization thus supporting provenance-based presentation for information only (e.g., [95, 170]) or they provide interactive visualizations and give users the possibility to customize their view of provenance (e.g., [120, 124]).

For the presentation application, expert users (experts on the production process and the end product) are typically responsible for capture while consumers of provenance are a more general, not necessarily expert public, for which adequate visualizations and interactions need to be developed.

**Attribution.** *Attribution* [166] limits understandability to one particular aspect of the production process, i.e., who / what authored (which part of) it. This information is useful to establish copyright and ownership of data, enable its citation, or determine liability for erroneous data. This application falls under the scope of communication between users since it clarifies actions performed by each collaborator.

Attribution has been considered in various provenance-enabled systems. PASS [142] supports attribution at the level of command line executions where the meta-data (= provenance) records user and group information for particular command line executions. Provenance is extended in [4, 118, 123, 168] to further describe the execution environment (operating system, user session name etc.). On cloud infrastructures, tracking files transferred from one machine to another is a challenge, addressed for instance in [170] by collecting information about senders and receivers.

The systems mentioned so far record and make accessible meta-data that is explicitly available (e.g., through user management). However, it is also possible that attribution needs to be derived as it is implicitly hidden in the data. As an example, [63] reconstructs the provenance of data through hidden dependencies between social data. This includes identifying users adapting another user's message, without explicitly referring to it.

## 2.2 Reproducibility

Following [137], the reproducibility of a result consists of starting with the same materials and methods and checking if a prior result can be confirmed. As provenance is, in its generality, capable of recording anything happening during processing, it has naturally been used for reproducibility.

We divide reproducibility ino two sub-classes, depending on which entities produce and consume the provenance.

**Recall.** Recalling one's steps in obtaining a result is essential to reproduce one's work. Collecting provenance recording actions and processing supports users for this application of *recall*. Typically, this *aide memoire* is targeted towards the entity responsible for the process, so provenance producer and consumer are the same. Fig. 2 indicates this with the *self-self* annotation below the application class. Provenance

[1] http://git-scm.com

[2] https://www.mercurial-scm.org

for recall may for instance include what intermediate steps are part of the process, which changes were made (e.g., to fix bugs), how parameters were set and how they were varied, etc. Remembering past actions on the process is important, e.g., to speed up the development process.

As stated in [162], recall is beneficial especially over extended or intermittent periods of analysis and [121] demonstrates the importance of supporting recollection, as even over a short time span, participants are unable to accurately remember the steps in their analysis process.

Recall is rarely considered as sole application of provenance. Supporting recall is typically coupled with supporting further applications. These "secondary" applications focus on why users want to recall what they did previously. We decided to keep recall as an individual application for two reasons. First, supporting recall is at the core of supporting further applications and it covers the *how* recall is achieved as opposed to *why* it is needed. Second, the user-centric property of recall (decoupled from further applications) gives the opportunity to develop dedicated solutions in the future that potentially bear less overhead (due to secondary application) than existing ones.

**Replication.** *Replication* targets reproducibility of a final result not only by its original author but a wider group of experts, typically working in the same domain [162]. In this context, captured provenance needs to be rich enough to allow other experts to reproduce the results.

Example applications leveraging provenance for replicating results include [18, 37, 50, 126, 137, 157]. These applications are very diverse with respect to the use cases and platforms they support, e.g., Big Data processing using the Pig language and running on Hadoop Map Reduce clusters [116], grid-based scientific workflows [181], scientific experiments in various contexts [50, 88, 169], visualizations [18], system configurations [123] (e.g., DHCP configuration, setting up SSH connections as reported in [124]), or testing software programs submitted jointly with scientific publications to validate reported results [157]. Not only does the provenance collected allow one to reproduce the obtained results, but also analyzing this provenance for different versions of a process allows to study the impact and effect of differences between the versions or predict the behavior of the system as changes to the transformation occur.

### 2.3 Quality

This section summarizes the use of provenance to either improve the quality of the end product or the production process. Whereas improving the process is typically done by experts on the process, data quality issues may be tackled by any individual handling the data and who is not necessarily knowledgable in the way the data studied was obtained. In this case, provenance may shed some light on quality issues in data (e.g., while not knowing what an expert user has done, noticing quality issues for which the provenance reveals that they are all caused by the same expert possibly helps finding and fixing the cause of the issue).

**Process Quality.** In general, process quality focuses on assessing and possibly improving the quality of a process. Quality has many dimensions, e.g., it includes correctness, performance, or fault-tolerance. In this context, provenance is commonly used for monitoring and debugging applications to assess and improve certain quality dimensions.

The processes considered in the literature are very diverse: [129] uses provenance of Web service states (and state changes) to support developers in selecting adequate services for their particular application. Opposed to that, [164] focuses on improving the entry set of dictionaries used during information extraction to better label entities in unstructured text. Works [113, 123, 140, 141, 156, 170] collect provenance for distributed systems (cloud computing) to detect cloud security problems such as malicious actions and data policy violations. For Map/Reduce programs, [3] collects provenance to decide how to optimize future executions by deciding where to store related data blocks that might be processed jointly.

In the context of scientific workflows, finding related workflows to the one that is being designed may help speed-up the development and improve the quality of these workflows. Provenance traces have been explored in assessing relatedness [18, 78]. Provenance has further been used to debug SQL queries [90, 95, 139], transactions [16], schema mappings [51, 81, 111], or preference queries [92]. Finally, provenance also supports improving the fault-tolerance of scientific workflow executions [55, 115] and making fault-tolerance protocols more robust [7].

**Data Quality.** Data quality encompasses various dimensions and metrics to assess the quality of data, including completeness, accuracy, timeliness, or trust [19, 182]. Clearly, all methods improving process quality to obtain a higher quality end result belong to this class as well. In addition, provenance is often considered when it comes to either assess the quality of data sources and providers (e.g., [60, 91, 168]) or result data based on the sources it has been derived from (e.g., [60, 86, 117, 186] consider trust, [145] considers integrity and accuracy).

## 3 Data provenance

We now start our discussion of provenance types, starting with data provenance (the most specific type of provenance as shown in Fig. 1). Data provenance counts more than two decades of research [183, 185], which we classify according to the hierarchy depicted in Fig. 4.

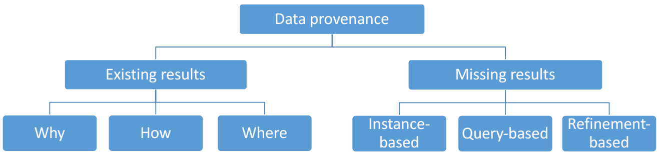

Fig. 4: Classification of data provenance research

The first level includes two classes: the first class, provenance of *existing results*, focuses on describing both the origins and the processing of data in the result of a query. Opposed to that, the second first-level class considers possible origins and processing steps causing expected data to be missing from a query result. It is therefore labeled as provenance of *missing results* (aka why-not provenance [43]).

The second level of data provenance of existing results reproduces the original classification of [46]. It classifies research along three main lines based on *where* [33] the data were copied from (exactly what attributes of which tuples), *why* [33, 58] a query answer was produced (i.e., based on what source data), and *how* [85] data were manipulated by the query to produce the result data in question. Considering the provenance of missing results, we consolidate previously used classifications [23,98]. Having the common goal to explain why some data are not part of a query result, the output is often referred to as explanations. These may take several forms: *instance-based* explanations identify source data that is missing from the sources to produce the desired result, whereas *query-based* explanations identify query operators that “lose” relevant data along the way. *Refinement-based* explanations tell how queries would need to be adapted to get the desired result.

We cover each class in more detail in the following. Sec. 3.1 focuses on provenance of existing results whereas Sec. 3.2 covers provenance of missing results. Each discussion first considers the general input, output, and processes supported before we delve into specializations of these.

## 3.1 Provenance of existing results

Due to space constraints, this section briefly summarizes research defining the different types of provenance of existing results and we refer readers to [46] for further details. In addition, we cover concepts relevant for our novel detailed survey on the provenance of missing results.

**Input.** As indicated in Sec. 1.2, data provenance restricts the process to a query with clearly defined semantics. In general, we denote the query as $Q$. The input further includes input data $D$, its data model being compatible with $Q$ (e.g., relational for SQL, XML for XQuery, RDF for SPARQL). Data provenance computation further requires the result of executing $Q$ over $D$, denoted $Q(D)$.

**Output.** In general, methods capturing data provenance compute a specific type of data provenance (as further discussed below) for a set $R \subseteq Q(D)$, where $R = \{t_{R_1}, \ldots, t_{R_n}\}$. Independently of whether an approach computes why, how, or where provenance, data provenance allows to obtain the provenance for each individual data item (e.g., relational tuple or XML element) $t_{R_i} \in R$. Note that given the fact that the foundations of all provenance types have been first defined and analyzed on relational data, we will often say tuple instead of data item.

**Queries.** The query languages expressing $Q$ vary between proposed solutions. $Q$ may be a query expressed in query languages such as SQL [58], Datalog [85], XQuery [73], or SPARQL [177]. Further formalisms with clear semantics on how each tupte $t_i \in D$ is processed also support the collection of data provenance, e.g., schema mappings [81].

### *3.1.1 Why-provenance*

Why-provenance identifies, given $Q$, $D$, $Q(D)$ and $R$, which tuples $DP \subseteq D$ are involved in producing the tuples of $R$. One of the first works formalizing this notion defines the *lineage* of data in the output of (materialized) relational views in a warehouse environment [57, 58]. The solution covers a large class of queries (SQL queries with selection, projection, join, aggregation, union, set difference, and further operators typical in data warehouses). While this approach guarantees that the returned set of lineage (source) tuples is sufficient to produce $R$, not all returned lineage tuples may be necessary simultaneously. To overcome this shortcoming, [33] introduces the notion of witness basis. Its computation involves considering the structure of the query. As a very simple example to distinguish lineage from why-provenance based on a witness basis, consider a single result tuple $t_R$ of a query over a relational database, and $Q = (R \bowtie S) \cup (R \bowtie T)$. Assuming tables $R = \{t_1\}$, $S = \{t_2\}$, and $T = \{t_3\}$, a returned why-provenance based on a witness basis of $\{\{t_1, t_2\}, \{t_1, t_3\}\}$ tells us that $t_R$ can be obtained based either on $t_1$ and $t_2$, or $t_1$ and $t_3$. Opposed to that, lineage returns $\{t_1, t_2, t_3\}$, leaving open if all three tuples are simultaneously necessary or not.

Given that the witness basis considers the structure of the query, provenance may not be invariant with respect to equivalent query rewritings. This means that in general, the computed provenance may be different for equivalent queries. The minimal witness basis [33] allows to guarantee this property for conjunctive queries with equalities only (both for relational and nested data).

### *3.1.2 How-provenance*

In addition to identifying which tuples contributed to a result (as why-provenance does), how-provenance provides a semiring-based tuple annotation for each result tuple $t_R \in R$ that encodes how source tuples combine to produce $t_R$ [110]. The work of [85] has led to the definition of provenance using polynomials, a special class of semiring. The model applies to various database semantics, including relational, incomplete, or probabilistic databases. Reusing the same sample query $Q$ over relations $R$, $S$, and $T$ as above, a provenance polynomial for a result tuple $t_R$ may be $t_1 * t_2 + t_1 * t_3$. This polynomial indicates that $t_R$ can be either obtained by joining source tuple $t_1$ with source tuple $t_2$ (both tuples are required) or, it can be obtained by joining $t_1$ with $t_3$. Originally defined for unions of conjunctive queries, the framework has been applied to nested XML data [73] and RDF [177] (together with why- and where-provenance), adapted to queries with aggregation [11] (moving from semirings to semi-modules), and studied for queries with relational difference [10], showing that in general, the semiring framework cannot be extended while satisfying expected equivalence axioms.

### *3.1.3 Where-provenance*

Whereas why- and how-provenance can be considered as annotating a result tuple $t_R$ with provenance involving tuples from $D$, where-provenance aims at identifying where data of $t_R$ (e.g., an attribute value) were copied from [33]. Hence, the provenance refers to specific locations, e.g., specific cells of a relational table. For instance, consider a query $Q = \Pi_{R.B} R \bowtie S$ of relations $R(A, B)$ and $S(B, C)$ and joining tuples $t_1$ and $t_2$, respectively. Then, the where-provenance of the result tuple $t_R$ will always refer to the attribute value of $B$ in relation $R$, denoted $R.B$. Opposed to that, the why-provenance $\{t_1, t_2\}$ and the how-provenance $t_1 * t_2$ of $t_R$ do not encode whether the result value of $R.B$ was copied from $R.B$ or $S.B$.

An alternative view of where-provenance can be obtained by annotating attribute values in $D$ and then propagating these annotations through the processing of $Q$ all the way to the result. Then, a result value can be associated to its original location through the propagated annotation [22,34].

## 3.2 Provenance of missing results

Research on provenance of missing results aims at explaining why some expected data are not part of a query result. Fig. 4 divides provenance of missing results depending on whether a solution produces instance-based, query-based, or refinement-based explanations.

Analogously to the previous subsection, we structure the discussion by first introducing the general input, output, and query languages being supported by the individual methods. We then discuss details on each type of explanation. The next subsections discuss algorithms producing explanations for missing query results for each explanation class.

**Input.** The input to algorithms explaining missing results, as defined by the framework of [94] generally consists of a 5-tuple $(T_R, Q, Q(D), D, C)$, where $T_R = \{t_{R_1}, \ldots, t_{R_n}\}$ is a set of conditional tuples [105] describing the missing results, $Q = \{q_1, \ldots, q_n\}$ is a set of queries, $Q(D) = \{q_1(D), \ldots, q_n(D)\}$ are the results of these queries over a source instance $D$, and $C$ is a set of constraints defined over the remaining four components of the 5-tuple. As the detailed discussion of individual methods below shows, most methods limit $Q$ (and consequently $Q(D)$) to one query only. We further distinguish two types of *input questions*: most methods support what we call *simple why-not questions*, where any tuple in $T_R$ either specifies missing results by comparing their attribute values to constants only or join conditions (i.e., comparisons between attribute values) only refer to attributes originating from the same relation $R \in D$. In all other cases, the conditional tuples in $T_R$ are qualified as *complex why-not questions*, e.g., if the missing tuple $(a, c)$ expresses a why-not question over the result of a query $q = \Pi_{R.A,S.C}(R \bowtie S)$ . Finally, the constraints expressed in $C$ by different methods highly vary.

**Output.** The output, i.e., the *explanation* returned by a method strongly depends on the class of explanation (instance-based, query-based, or refinement-based). Instance-based methods typically return changes to the input database $D$ that, if actually performed, would yield query results that satisfy all conditions expressed in $T_R$ while satisfying any additional constraints defined in $C$. Source instance edit operations include tuple insertion, tuple deletion, and value update in existing tuples. Opposed to that, query-based approaches return, in different forms (i.e., sets or polynomials), query conditions (joins and selections) that prune data present in $D$ and that may contribute to missing results as defined in $T_R$. Finally, refinement-based approaches may modify an original input setting $(T_R, Q, Q(D), D, C)$ to $(T'_R, Q', Q'(D), D, C)$ such that $T'_R \in Q'(D)$ while satisfying the constraints $C$.

**Queries.** For methods focusing on relational queries (using relational algebra, Datalog, or SQL), we distinguish between conjunctive queries (with and without inequalities), also called select-project-join (SPJ) queries, queries involving set union, queries involving aggregation and grouping, and non-monotonous queries with set difference. Further queries, such as (reverse) skyline queries or (reverse) top-k queries have been studied as well.

Given the above input, output, and query characteristics, we now survey specific algorithms, which are summarized in Tab. 1 (relational queries) and Tab. 2 (other queries).

| Method characteristics | | | Explanation based on | | | Queries | | | |
|---|---|---|---|---|---|---|---|---|---|
| *name* | *input question* | *explanation* | *instance* | *query* | *refinement* | *SPJ* | *Union* | *Aggregation* | *Difference* |
| Missing-Answers [101] | simple | source edits (insert / update) | ✓ | | | ✓ | | | |
| Artemis [97] | complex | source table edits (insert) | ✓ | | | ✓ | ✓ | ✓ | |
| PGames [114, 163] | simple | source table edits (insert) | ✓ | | | ✓ | ✓ | | ✓ |
| Meliou *et. al.* [128] | simple | causes (tuples) and responsibility | ✓ | | | ✓ | | | |
| Why-Not [43] | simple | query operators | | ✓ | | ✓ | ✓ | | |
| NedExplain [25] | simple | query operators | | ✓ | | ✓ | ✓ | ✓ | |
| TED [23, 24] | complex | query operator polynomials | | ✓ | | ✓ | ✓ | | |
| Conseil [94] | simple | source edits (insert/delete) + operators | ✓ | ✓ | | ✓ | ✓ | ✓ | ✓ |
| ConQuer [179] | complex | refined query | | | ✓ | ✓ | | ✓ | |
| FlexIQ [107] | simple | refined query and why-not question | | | ✓ | ✓ | | | |
| EFQ [26] | simple | refined query | | | ✓ | ✓ | | | |
| TALOS [180] | complex | refined query | | | ✓ | ✓ | | ✓ | |

Table 1: Main characteristics of algorithms producing explanations of results missing from the result of a relational query

| Algorithm | Explanation | Query |
|---|---|---|
| **Instance-based explanations** | | |
| Calv. *et. al.* [38] | additions to ABox | instance & conj. queries<br>over DL-Lite ontology |
| ten Cate *et. al.* [176] | ontology concepts | conj. queries with comparisons |
| **Refinement-based explanations** | | |
| He and Lo [92] | $q' = (k', W')$ | top-k query $q = (k, W)$ |
| He and Lo [93] | $q' = (k', W), T'_R$ | top-k (dominating) query |
| WQRTQ [77] | $q' = (k', W, t'), T'_R$ | reverse top-k query $q = (k, W, t)$ |
| Chester [48] | $T'_R$ and constraints | constrained skyline query |
| Islam *et. al.* [108] | $t', T'_R$ | reverse skyline for query tuple $t$ |

Table 2: Summary of solutions providing explanations of missing results for non-relational queries. All solutions support simple why-not questions. For refinement-based explanations, returned explanations may refine both (components of) the original query $q$ to $q'$ and components of the why-not input $(T_R, \{q\}, \{q(D)\}, D, C)$.

### 3.2.1 Instance-Based Explanations

**Relational queries.** Missing-Answers (MA) [101] computes instance-based explanations that either insert or update the source data for a single missing result and a single SQL query including selection, projection, and join. To limit the number of returned explanations, trust constraints allow to specify that certain tables or attributes cannot be modified, thus preventing tuple insertions and value updates on these. Overall, the input 5-tuple for the MA algorithm is $(\{t_R\}, \{q\}, \{q(D)\}, D, \{R, A\})$, with $R$ being the set of trusted relations and $A$ the set of trusted attributes, and $t_R$ being a simple why-not question. As example, consider $q = R \bowtie S$ with $R = \{t_1\}$ and $S = \emptyset$. Clearly, the result of the query is empty and one may ask the question why we do not have any tuple (no matter the actual values), i.e., $t_R = [v_1, v_2, v_3]$, where each $v_i$ is a variable on which no further constraint applies. One possible instance-based explanation is the insertion of a tuple $[v_2, v_3]$ into $S$ such that $v_2$ equals the joined value of $t_1$. Alternatively, inserting a pair of tuples that join together, i.e., $[v_1, v_2], [v_2, v_3]$ into $R$ and $S$, respectively, would also yield a tuple satisfying the template given by $t_R$.

Artemis [97] generalizes MA to SQL queries involving selection, projection, join, union, or aggregation with grouping. Furthermore, it supports complex why-not questions consisting of possibly more than one conditional tuple over a set of queries (views) over a source instance $D$. On the other hand, Artemis limits to instance-based explanations that insert new tuples to $D$ (i.e., updates to existing tuples are not considered). Similarly to the trust constraints defined by MA, Artemis allows to specify a set $Q_{im}$ of immutable views where no data should be added. These views may either be the sources (hence preventing insertions to $D$) or views of part of $Q$ that should not be affected by source insertions, thus avoiding undesired side-effects of (potential) source insertions on query results. In addition to preventing such side-effects, Artemis allows to specify that side-effects should be minimized (in case they cannot be completely avoided) for given views $Q_m$. In summary, Artemis supports the input $(T_R, Q, Q(D), D, \{Q_{im}, Q_m\})$, $T_R$ including complex why-not questions. By leveraging the minimal witness basis [33], guarantees on the minimality of explanations for conjunctive queries can be given.

Whereas the previous approaches specialize on explaining missing answers only, the following two methods actually aim at unifying both provenance of existing results and instance-based provenance of missing results.

Provenance games (PGames) [114, 163] do so by modelling either a single existing tuple or a single missing tuple as a won or lost game, respectively. By modelling the query as a game, PGames are then able to retrace the different moves that lead to winning or losing the game (and the result tuple). Essentially, winning paths provide how-provenance equivalent to the semiring provenance [85] of existing results whereas paths that lead to a loss translate to an instance based explanation that describes which tuples need to be inserted in $D$ in order to obtain the missing result. The proposed method focuses on non recursive Datalog and supports first oder queries. As the model considers a single (missing) tuple and a single query without any further constraints, the input characteristics summarize as $(\{t_R\}, \{q\}, \{q(D)\}, D, \emptyset)$.

Another approach unifying provenance of existing results and missing results leverages the concepts of causality and responsibility [128]. Essentially, this methods relies on the identification of a set of tuples $D^* \subseteq D$ that either corresponds to the why-provenance of an existing result tuple $t \in Q(D)$ or the instance-based explanation of a missing result $t_R$ as determined for instance using [101] to identify causes and quantify their responsibility in contributing to the (lack of) result tuple. More specifically, given $D^*$, a tuple $t_i \in D^*$ is a cause of $t_R$ with a certain responsibility depending on its contingency set. The contingency set of $t_i$ includes all tuples in $D^*$ that should be updated along with $t_i$ to obtain $t_R$ in the query result. The results apply to conjunctive queries and simple why-not questions. Within our framework, the input characteristics summarize as $(\{t_R\}, \{q\}, \{q(D)\}, D, \emptyset)$.

**Ontologies.** Whereas the methods discussed above focus on relational queries, the following methods consider explaining missing results in the presence of ontologies.

Calvanese et al. [38] study the problem of why-not provenance for instance and (unions of) conjunctive queries over description logic ontologies in $DL - Lite_{\mathcal{A}}$. The theoretical formalization and study leverage abductive reasoning. In summary, given a query $q$, a consistent ontology $\mathcal{O}$ and a missing tuple $t_R$, an instance-based explanation takes the form of additions to the ABox that result in $t_R$ becoming an answer while preserving the consistency of $\mathcal{O}$.

While the work above considers explaining missing results over an ontology, ten Cate et al. [176] have studied the problem of how to leverage an ontology to explain a missing query result tuple. In this work, explanations generalize on the previous notion of instance-based explanation as the returned explanations are composed of concepts rather than data of the extension. More precisely, an explanation of a missing tuple $t_R$ is a tuple of concepts from the ontology whose extension includes $t_R$ while not including any tuples from the query result $q(D)$. The proposed framework focuses on conjunctive queries with comparisons over a database schema with a related ontology and supports simple why-not questions.

While instance-based explanations are typically considered user-friendly as they explain missing answers based on examples from the data, all approaches suffer from the inherent complexity of computing instance-based explanations that comes from the fact that in general, a solution may have to enumerate all possible explanations. Indeed, the number of explanations grows exponentially (in $O(N^J)$, where $N$ is the largest size of a relation in the schema of $q$ and $J$ the number of joins in $Q$). In contrast to instance-based explanations, query-based explanations, covered next, return a much more concise explanation that can typically also be computed more efficiently.

### *3.2.2 Query-based explanations*

While instance-based explanations explain the missing result $t_R$ based on data, query-based explanations comprise query operators potentially being responsible for eliminating a missing result from $q(D)$, implicitly assuming that $D$ is correct and complete. In the previously given example, a simple query-based explanation returns the join as culprit operator, as it is too strict to return any result given $D$.

A first approach returning query-based explanations is Why-Not [43]. While being motivated by providing why-not provenance for scientific workflows modelled as directed acyclic graphs, the solution applies to graphs where nodes are operators from relational algebra (in particular, selection projection, join, and union). In this setting, it takes as input a tuple $t_R$ missing from the result $q(D)$ of a relational query $q$ in the form of an algebraic query tree. First, it identifies tuples in $D$ that contain the constant values or that satisfy the conditions of the simple missing-answer as specified by $t_R$ while not being part of the lineage (as defined in [58]) of any result tuple. These tuples, named *compatible tuples*, are then traced over the query operators to identify which operators have them as input but not as output.

NedExplain [25] takes a similar approach to Why-Not as it also traces tuples identified in the source relations over a query tree representation of a relational query. However, these compatible tuples are not restricted to source tuples absent in the lineage of any result tuple. Based on this modified base assumption and a novel formal definition of query-based explanations, NedExplain computes a generally more complete, correct, and detailed set of explanations than Why-Not. In addition, it supports queries involving selection, projection, join, and aggregation (SPJA queries) as well as unions of such SPJA queries.

An approach combining both query-based and instance-based explanation has been proposed in [94]. It presents both a unified framework for these two types of explanations as well as an algorithm, named Conseil. Given a single missing tuple and a relational query involving selection, projection, union, aggregation, and a single set difference (restriction for both complexity and usability reasons) represented as a query tree, Conseil returns hybrid explanations that consist of two components (an instance-based component and a query-based component). Intuitively, the explanations describe in their query-based component what query operators would be responsible for pruning compatible tuples and hence the missing result should the source table modifications described in the instance-based component be performed.

As the above approaches consider the query as a tree of relational operators, the explanations returned are not invariant with respect to equivalent query rewritings. To remedy to this problem, TED [24] proposes a framework that returns equivalent query-based explanations for equivalent conjunctive queries. Explanations take the form of polynomials in analogy to how-provenance. Whereas the annotations of the how-provenance semirings describe combinations of tuples, the explanations returned by TED annotate the query and describe combinations of query operators that prune combinations of compatible tuples that may otherwise lead to the missing tuple. For instance, an explanation for why a tuple $t_R$ is not in the result of a query $q = \sigma_{A=5}(R \bowtie S)$ may read (in a simplified way for illustration and conciseness) $5 * \sigma + 10 * \bowtie + 3 * \sigma * \bowtie$, meaning that among all compatible tuples combined from source relations $R$ and $S$, 5 would satisfy the join but are pruned by the selection, 10 satisfy the selection but not the join, and 3 neither satisfy the selection nor the join. How to efficiently compute these query-based explanations for unions of conjunctive queries and complex why-not questions is discussed in [23].

The worst-case size of a query-based explanation varies between the number of conditions in $q$, denoted $cond(q)$ for Why-Not and NedExplain and $2^{cond(q)}$ for TED. The runtime of the algorithms tracing compatible data through the query tree is dominated by the time needed to iterate over intermediate query operator outputs of worst size $N_O$ (at most once for every node in the logical tree of $q$) to check whether a compatible tuple passed an operator. As for TED, its complexity is in $O(N^{|from(q)|})$, where $N$ is the maximum size of a source relation in the set of relations referred to in the FROM clause of $q$, denoted $from(q)$.

### 3.2.3 Refinement-based explanations

Refinement-based explanations encode how to modify a query $q$ or the specified missing results $T_R$ into a refined query $q'$ or missing results $T'_R$ such that all tuples of $T'_R \in q'(D)$. In the relational context and reusing the same example as previously, the query $q$ may have to be rewritten by replacing the join between $R$ and $S$ by a left outer join to yield a result tuple.

**Relational queries.** ConQuer [179] returns rewritings of an initial relational query (including selection, projection, join, and aggregation) for possibly multiple missing result tuples $T_R$. In addition, it allows to specify constraints spanning multiple missing result tuples. In our general framework, this can be modelled via the set of constraints $\mathcal{C}$. The returned refined queries all return the missing tuples in their result while retaining all query results of the original query $q$ and possibly minimizing the appearance of side-effect tuples not in $T_R \cup q(D)$. Changes to the query apply on the conditions of the WHERE clause, if these are not sufficient, ConQuer further considers changes to the query schema. To reduce the number of side-effect tuples, the candidate queries returned at this stage are further constrained with additional conditions in the WHERE-clause. The refined queries ConQuer identifies are ranked based on their similarity to $q$ and the number of side-effects. A key concept that Conquer relies on for both identifying selection predicates and ranking queries is skyline computation [29].

Another approach relying on skyline computations is FlexIQ [107] that aims at resolving both existing but unexpected as well as missing but expected query results. Both these types of tuples are specified as part of the set of input tuples. Focusing on SPJ queries, FlexIQ computes refinements of $q$ such that the query result of these refinements include all expected but missing tuples, removes all unexpected tuples and retains all remaining tuples from the initial query result $q(D)$. When no exact query refinement can be found, the algorithm suggests an approximate solution by refining the set of missing tuples.

EFQ [26] presents a framework producing rewritings of SPJ queries to include a missing result $t_R$ in the result of a given query $q$. While being similar to ConQuer, it reduces the search space of rewritings by leveraging previously computed why-not provenance polynomials [23]. In addition, it also considers rewriting joins to outer joins.

Opposed to the above methods relying on skyline computations, TALOS [180] relies on decision trees. TALOS has originally been developed to solve the problem of generating instance equivalent queries that, given a set of result tuples $T$ and optionally an initial query $q$, generate the same output set $T$ if executed over the same input dataset. By setting $T$ to include all results of a query and missing tuples, i.e., $T = q(D) \cup T_R$, this method can be used to produce refinement-based explanations for missing tuples. The solution proposed focuses on queries with selection, projection, join, and aggregation and supports complex why-not questions. The changes applied to the initial query are analogous to those described for ConQuer.

**(Reverse) top-k queries.** Refinement-based explanations have been considered beyond relational queries and we begin the discussion of these techniques with solutions for answering why-not questions for top-k queries and reverse top-k queries. In general, a top-k query is a query $q = (k, W)$ for the $k$ best tuples in a dataset based on a ranking function assigning the weights in $W$ to corresponding relation attributes. In this context, He and Lo [92] have studied refining $q$ by queries $q' = (k', W')$ to answer the question why a set of missing tuples $T_R$ are not part of the result of $q$. Intuitively, the refined query $q'$ includes all tuples in $T_R$ in its top-k' results. Note that the refined queries are not required to retain all results (or a subset thereof) of $q(D)$. As solving the exact problem is computationally prohibitive, the authors propose an approximate algorithm based on sampling of weighting vectors and that is optimized to avoid unnecessary computations of refined queries, identified as queries dominated by previously computed queries. These methods are adapted to top-k dominating queries [93] that return the top-k tuples based on the number of tuples that they dominate. Again, the why-not question asks for a set of tuples $T_R$ not in the result of $q$. However, the rewritten queries refine $k$ to $k'$ and possibly also $T_R$ to $T'_R$.

We now discuss query refinement for answering why-not questions on reverse top-k queries. Given a set of tuples and a set of preference vectors, a reverse top-k query returns, for a particular tuple $t$, the set of preference vectors that contain $t$ in their top-k result. A why-not question over the result of a reverse skyline query expresses why certain preference vectors $T_R$ are missing from the result of a reverse top-k query. WQRTQ [77] proposes a framework to answer such why-not questions by refining $T_R$ (i.e., the preference vectors of the why-not question), the tuple $t$, or $k$. To find an optimal solution, methods minimizing a penalty score that measures the difference of the proposed rewriting to the original setting are proposed.

**(Reverse) skyline queries.** As final type of query for which answering why-not questions has been considered, we summarize approaches focusing on skyline queries and reverse skyline queries. In general, a skyline query returns all tuples that are not dominated by any other tuples. One variant of skyline queries named constrained skyline queries [155] allow further constraints on the value range of attributes. For such constrained skyline queries, Chester and Assent [48] propose refinement-based explanations for why-not questions that ask why a certain tuple $t_R$ is not in the skyline returned by a skyline query. The refinement focuses on modifying the constraints of the constrained skyline query and may extend to refining $t_R$.

The definition of reverse skyline queries is based on the definition of dynamic skyline queries [155]. Essentially, the dynamic skyline for a given query tuple $t$ includes all tuples that are not dominated by other tuples according to their distance to the query tuple $t$. Then, the reverse skyline query [66] of a query tuple $t$ includes all tuples that have $t$ in their dynamic skyline. In this context, the why-not question as defined in [108] asks why a certain tuple $t_R$ is missing from the reverse skyline of a query tuple $t$. As refinements, the algorithm proposed in [108] returns refinements to $t$ or $t_R$ such that the refined missing tuple $t'_R$ appears in the reverse skyline of the refined query tuple $t'$ and that all tuples previously returned as part of the reverse skyline are retained in the result of the refined query.

While we can generally say that the number of explanations returned by refinement-based approaches is exponential in the size of updatable input fragments, we cannot make general comments on the computational complexity as the discussed solutions employ different algorithms and optimizations for different queries. A more detailed discussion would require significantly more space and is thus out of the scope of this paper.

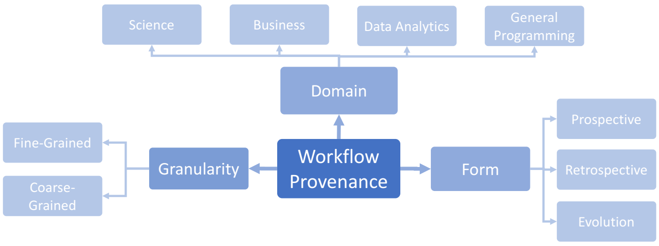

Fig. 5: Workflow Provenance Overview

## 4 Workflow provenance

Fig. 5 distinguishes work on workflow provenance, which has emerged more than ten years ago [6,20,150], along three dimensions based on their *domain* of application, its *granularity*, and the *form* of captured provenance.

We identify four major domains: business, science, data analytics, and general programming. While the first two domains are adopted from previous surveys on workflow provenance [74, 136, 162], the latter ones reflect recent advancements in workflow provenance research. Concerning the granularity of captured provenance, it may vary between coarse-grained and fine-grained provenance. The form of provenance is the third dimension we consider. In this survey, we focus on work made since [74] in the field of retrospective, prospective provenance, and evolution provenance.

Before delving into the details of the individual dimensions, we describe the general input, output, and processes (i.e., workflows) covered by worfklow provenance.

**Input.** Workflow provenance solutions commonly obtain provenance based on the input $I = (W, D, C)$: $W$ represents the workflow definition in form of a directed graph, i.e., $W = (M, P)$ where $M$ is the set of nodes representing workflow modules while the edges $P$ describe dependencies between the modules. $D$ refers to the input data (could be structured relational data, nested semi-structured data or even unstructured data). In order to process $D$, (modules of) the workflow $W$ may require execution parameters, which are input through the execution context $C$.

**Output.** Depending on the application of the provenance, the level of available instrumentation, and further input properties, the form and granularity of output provenance may vary. For instance, it may be fine-grained, i.e., at the level of individual data items processed by a workflow, which is similar to the output of data provenance solutions of existing results. Opposed to that, the output may limit to modifications made on the workflow definition, parameters, or other input. We discuss the different forms and granularities of the output further below.

**Workflows.** The general graph model for a workflow introduced above covers different, more constrained graph models, which, as we will see, coincide with workflow types

| Solution | Granularity | | Form | | |
|---|---|---|---|---|---|
| | Coarse | Fine | Prospective | Retrospective | Evolution |
| **Science** | | | | | |
| DFL [1] | ✓ | ✓ | | ✓ | |
| Galaxy [84] | ✓ | ✓ | | ✓ | |
| Kepler [6,32,126] | ✓ | ✓ | ✓ | ✓ | ✓ |
| Taverna [5,131,135] | ✓ | ✓ | ✓ | ✓ | |
| VisTrails [20,37,75] | ✓ | | | ✓ | ✓ |
| WebLab PROV [8] | | ✓ | | ✓ | |
| **Business** | | | | | |
| [59,125,171] | ✓ | | | ✓ | |
| [172] | ✓ | | | ✓ | |
| **Data Analytics** | | | | | |
| Ariadne [82] | ✓ | ✓ | | ✓ | |
| Differential DF [52] | ✓ | ✓ | | ✓ | |
| HadoopProv [3] | ✓ | ✓ | | ✓ | |
| Inspector Gadget [153] | ✓ | ✓ | | ✓ | |
| Lipstick [9] | ✓ | ✓ | | ✓ | |
| Newt [122] | ✓ | ✓ | | ✓ | |
| RAMP [104] | ✓ | ✓ | | ✓ | |
| Titian [106] | ✓ | ✓ | | ✓ | |
| **General Programming** | | | | | |
| Jif [144] | ✓ | ✓ | ✓ | ✓ | |
| LLVM/SPADE [79,175] | ✓ | | | ✓ | |
| NoWorkflow [143,158] | | ✓ | | ✓ | |
| No+YesWorkflow [69,159] | ✓ | ✓ | ✓ | ✓ | |
| Pimentel et al. [160] | ✓ | | | | ✓ |
| RDataTracker [120] | ✓ | | | ✓ | |
| Starflow [15] | ✓ | | ✓ | ✓ | |
| YesWorkflow [127] | ✓ | | ✓ | | |

Table 3: Overview of Workflow Provenance Solutions

used in different domains. For instance, depending on the domain, edges in $P$ may represent data or control dependencies, the graph may be restricted to acyclic graphs, etc. Based on the general graph model, let us further define a generic execution model, which we will reuse later on. In general, we assume that a workflow module $m \in M$ is a black-box where we have no knowledge about its semantics or performed computation. Then, $m$ requires input data $I_m$ and execution context $C_m$ to produce output $O_m$. The execution of a workflow yields an execution log or trace $T$. $T$ is a directed acyclic graph $T = (E, CD)$ where $E$ represents events occuring during workflow execution (i.e. activation of modules in the workflow) and $CD$ represents causal dependencies between these events (i.e. the temporal order of the activation of modules in $W$).

We now discuss the different dimensions depicted in Fig. 5. Tab. 3 summarizes the discussion. The focus of this section's discussion is on the type of provenance captured by workflow systems. Further aspects such as accessing, querying, or exploring provenance are out of the scope of this section and are topics further discussed in [74, 162].

### 4.1 Granularity

Provenance solutions generate outputs with different levels of details. The two ends of the spectrum are coarse-grained and fine-grained provenance [32, 136, 173], however, note that solutions may produce provenance between these two extremes. In this context, knowledge about the internal behavior of workflow modules plays an important role. When modules' internal behavior or semantics are unknown, the modules are referred to as "black boxes". Other modules may have a well specified and documented internal behavior and are called "white boxes".

**Coarse-grained provenance.** In the most general setup, $M$ consists of black box modules only. As a consequence, provenance solutions may - in general - not be capable to provide derivation information on individual data items processed by a workflow. They rather rely on the assumption that the output data of a module depends as a whole on its full input and context [5, 136, 173]. More formally, we qualify provenance as coarse-grained if the entire output $O_m$ of a module $m \in M$ depends on its input $I_m$ as a whole and the complete execution context $C_m$, and this for all modules in $M$. That is, we have $\forall m \in M : m \times I_m \times C_m \rightarrow O_m$. Coarse-grained provenance is not detailed enough to express the individual derivation paths of seperate data items $o \in O_m$. It is also referred to as provenance of the workflow in other publications [61]. In our later discussion of individual solutions, note that when not mentioned otherwise, the systems at least support coarse-grained provenance.

**Fine-grained provenance.** In the most specific setup, provenance solutions can provide the derivation process of individual data items. Typically, these solutions either exploit knowledge about the internal behavior of white box modules in the workflow or about processing properties. Some systems pipeline individual items through the workflow. The provenance solutions can exploit this behavior to provide provenance for individual data items. We call provenance at such a level of detail *fine-grained provenance* [8,122,158]. It reflects that an item $o \in O_m$ produced by a workflow module $m \in M$ may not depend on the complete input $I_m$ and entire context $C_m$, but only on subsets $I'_m \subseteq I_m$ and $C'_m \subseteq C_m$. The finest-grained provenance is obtained when, $\forall m \in M, \forall o \in O_m : m \times I'_m \times C'_m \rightarrow o$ and $I'_m$ and $C'_m$ are both minimal. A synonym for fine-grained is provenance of the data [61].

**Relationship to data provenance.** Even though single data items are tracked with fine-grained provenance, it is not necessarily the same as data provenance. According to our type hierarchy shown in Fig. 1, data provenance restricts to the most specific type of process where queries are based on operators with clearly defined semantics. Given our definition of fine-grained workflow provenance, it now becomes evident that the transition between data and fine-grained workflow provenance is smooth. Work on workflow provenance recognizing and working towards unifying these two types of provenance include [1, 9, 119]. However, these limit the considered workflow modules to modules mapping to nested relational calculus. Within workflow provenance, we cover the whole spectrum from the finest granularity resembling data provenance to coarse-grained workflow provenance.

## 4.2 Prospective, retrospective, and evolution provenance

In addition to exhibiting different provenance granularities, workflow provenance also differs in the form of the supported provenance, where we distinguish between prospective provenance, retrospective provenance, and evolution provenance. This reflects the common classification adopted for workflow provenance [74, 136, 143, 160]. All three forms are independent from one another. Hence, a provenance solution may support only one, two, or all three of them.

**Prospective provenance.** Prospective provenance captures the structure and static context of a workflow. It is independent of any workflow execution or input data. Hence, provenance solutions statically derive it from the workflow definition $W$ and pre-specified parameters part of $C$. That is, the input limits to $I = (W, \emptyset, C)$.

Across different domains, prospective provenance is used to provide an abstracted overview of a workflow [5,69]. Prospective provenance solutions can reveal control flow as well as coarse-grained dataflow dependencies.

**Retrospective provenance.** Retrospective provenance is associated with the information about a workflow execution, i.e., with information becoming available when actually running the workflow. This information includes facts about the execution of each workflow step as well as the runtime environment. Retrospective provenance also preserves information on the resources that are accessed or generated during execution. Formally, provenance solutions may access all input parameters $I = (W, D, C)$ to obtain retrospective provenance. A common and general approach to capture retrospective provenance involves the instrumentation of the workflow or its execution environment. The instrumentation allows for obtaining an execution trace $T$ during workflow execution. The trace $T$ often becomes an essential part of the output generated by the provenance solution.

**Evolution provenance.** Evolution provenance reflects the changes made between two versions of the input $I$ and $I'$. These changes may apply on $W$, $D$, or $C$. Whenever at least one of them is altered the provenance solution keeps track of those changes. Evolution provenance alleviates rapid iterations on various data, parameters, and workflow manipulations [6, 37]. This form of provenance is also called provenance of the (development) process in the domain of scientific workflows [133].

Even though presented as separate classes, provenance solutions may consider combining the different forms. For instance, the ProvOne standard proposal[3] considers all three forms of provenance. Prov2ONE generates as provenance a graph containing prospective provenance from the workflow definition and automatically adds retrospective provenance when available [161]. In the context of scientific workflows, all forms of provenance support the workflow developers. For instance, prospective provenance, especially in the form of annotations about module semantics, may help to understand the retrospective provenance of a workflow execution and ease collaboration as well as workflow and data exploration [5, 40]. Evolution provenance can accelerate the decision process about the correct module semantics since multiple workflow executions can be compared. In general programming, solutions exist that combine prospective and retrospective provenance [69, 159]. These solutions allow for answering provenance questions no solution supporting only one form of provenance is able to answer.

## 4.3 Application domains

Workflow provenance has been employed in a variety of domains like experimental science, business, data analytics, and general programming. As we will see shortly, the domains are tightly coupled with different sets of restrictions on the general workflow graph model we have introduced. Again, the defined classes should not be considered as excluding each other. In the following, we describe characteristics of workflows and provenance solutions in the different domains and link back to the discussion on the form and granularity of generated provenance.

### *4.3.1 Scientific workflows*

Scientific workflows are typically created, managed, and executed in scientific workflow management systems (SWfMS). They have emerged for multiple scientific applications. For instance, Galaxy, Taverna, and Kepler originate in research on biomedical subjects like genome research or life sciences [32, 84, 150]. WebLab is designed for multimedia information processing [8]. Pegasus provides examples on astrophysical research [65]. VisTrails collects provenance on the exploration and analysis of environmental, medical, or other scientific data [20, 37].

**Workflow characteristics.** Scientific workflows have initially been modelled in a data-driven way. Thus, the edges $P$ of the workflow graph describe data dependencies, leading to acyclic graphs [65, 131]. Some SWfMSs (e.g., [32]) also allow to model control flow to some extent (e.g., control-loops), so resulting workflows may contain cycles.

**Granularity.** Especially in the domain of scientific workflows, initial efforts focus on collecting coarse-grained provenance [6, 150]. For instance, Kepler, Galaxy and Taverna collect module execution time and status [6, 84, 131]. This is part of the collected coarse-grained provenance since it is linked to a module as a whole rather than to individual tuples. Similarly, module annotations contribute to the coarse-grained provenance information [5, 6]. Fine-grained

[3] http://vcvcomputing.com/provone/provone.html

provenance solutions for SWfMSs collect data dependencies between individual data items. For instance, WebLab PROV is capable to track individual items in nested XML artefacts [8]. Taverna and Kepler provide similar means to capture the derivation paths of single data items [32, 135, 151].

**Form.** Many provenance solutions in the scientific domain allow to collect retrospective provenance at any granularity [6, 8, 150]. Their output typically is a provenance graph that is based on the execution trace $T$. Further solutions focus on capturing prospective provenance as well. They may not only rely on static workflow analysis, but also on annotations [5]. Evolution provenance is, for instance, supported by Kepler [6] or Vistrails [75]. One goal of evolution provenance is an extension of the W3C PROV model [130]. Initial ambitions by Missier et al. have led to the D-Prov extension [133], which has evolved to the ProvOne standard proposal. Oliveira et al. show that it is possible to map provenance data from multiple SWfMSs to the ProvOne standard, indicating the feasibility of the standard [152].

**Overhead.** The evaluation of existing provenance solutions in the scientific workflow provenance domain has a clear focus on qualitative aspects. Regarding runtime or space overhead for collecting provenance, no actual performance numbers or complexity analysis are provided.

### *4.3.2 Business workflows*

Business workflows accompany and support processes between and within companies.

**Workflow characteristics.** Business workflows consist of multiple activities, tasks, or events that may involve human interaction. Unlike scientific workflows, they are driven by the control flow [136]. That is why the edges $P$ of a business workflow $W$ represent control flow. Data flow is typically not even explicitly modelled in business workflows.

**Granularity.** Business process solutions mostly collect provenance at the granularity of individual files [59, 171]. A major challenge in collecting provenance for business workflows is to identify collection points for provenance and to establish a generic provenance model [125]. It may even be necessary to infer data dependencies from the execution order of workflow modules [172]. As a consequence, the provenance solutions are often not able to track the origin and derivation path of individual data items within each file [59]. Tan et. al point that business workflows are often assembled from web services whose internals are not known. These services do not allow for tracking fine-grained provenance out of the box [172]. However, provenance solutions typically track information about the origin, history, and invocation of the web service [136].

**Form.** In the contex of business workflows, provenance is often used for auditing, regulatory compliance, accountability or trustworthiness. Thus, the provenance solutions typically focus on retrospective provenance [59, 172].

**Overhead.** In this domain, recording of provenance is often required for legal compliance, rendering provenance tracking mandatory. That may be one reason why the work cited in this section does not provide overhead measurements.

### *4.3.3 Data analytics*

A third group of systems are the data analytics engines. Even though data analytics programs are typically data driven as the name implies, they differ from the above workflow systems. They do not originate in interdisciplinary sciences like scientific workflow systems and business process management tools. Their original purpose is to process semi-structured or unstructured data from heterogenous sources. Popular representatives of data analytics engines include Hadoop [184], Hive [178], Pig [154], and Spark [187]. For some of these engines provenance extensions are developed to deeply understand execution results and debug programs [9, 104, 106]. Initial provenance research has covered especially low level map/reduce engines like Hadoop as RAMP, Newt, and HadoopProv show [3, 104, 122]. Since implementing low level chains of map-reduce jobs is rigid and yields a noticeable amount of custom code for recurring problems [154] higher level languages like Pig [154], Hive [178], and Spark [187] have emerged. They simplify the implementation of complex data analytics programs by providing high-level, declarative modules like join, merge, or filter with clear semantics. Provenance solutions like Titian [106], Lipstick [9], and Inspector Gadget [153] are tailored to these higher level languages. Some solutions are custom-made for special applications, e.g., Ariadne is designed to efficiently record provenance information for streaming data in data analytics engines [82].

**Workflow characteristics.** Similarly to scientific workflows, data analytics workflows are data-driven. Thus, the edges $P$ generally describe data dependencies between data manipulating modules $M$. Control dependencies like loops as known from business workflows do not belong the predefined repertoire of possible operations. The workflow graph $W$ itself is acyclic [154, 178, 187].

**Granularity.** Provenance solutions for data analytics systems provide fine-grained provenance, which for these systems subsumes coarse-grained provenance as well. They capture the derivation path of individual data items [104, 106]. Amsterdamer et al. focus explicitly on data items in nested data structures [9]. They describe a formal model for Pig Latin that combines fine-grained and coarse-grained provenance. It reflects the internal state of the workflow models as well as fine-grained data dependencies. Chothia et al. add the modification time for each item and module to provide exact provenance for workflows with non-monotonic modules [52].

**Form.** Similarly to scientific workflow solutions, solutions instrument data analytics systems to collect retrospective provenance during workflow execution. Lipstick, Titian, RAMP, Newt, and others collect fine-grained retrospective provenance [9, 104, 106, 122]. To the best of our knowledge, no work focusing on prospective or evolution provenance has been published in the domain of data analytics.

**Why- and how-provenance.** All solutions mentioned above allow to track individual data items. However, they do not capture how the manipulation of each module contributes to the manipulation. Therefore, all of the solutions support what we describe as why-provenance in Sec. 3. Only Amsterdamer et al. [9] and Chothia et al. [52] target how-provenance in the sense of Sec. 3. They store the provenance in a graph rather than provenance polynomials. It keeps track not only of the derivation path of a data item but also of the manipulation process each data item undergoes.

**Monotonicity limitations.** Most provenance solutions for data analytics provide accurate provenance only in deterministic workflows that contain one non-monotonic module at most [52]. A module $m$ is considered monotonic if for any input data $I'_m \subseteq I_m$ the constraint $O'_m \subseteq O_m$ holds [104]. While map-modules are inherently monotonic, many complex modules including reduce-modules may violate this constraint. Examples of violating modules include top-k modules [104], a stable-matching modules [52], or a reduce module that emits only those data items which exclusively occur in either of two input datasets [52]. Chothia et al. describe a provenance solution for analytics systems that allows for precise provenance tracking in workflows with more than one non-monotonic module. That is possible, because they record not only the derivation path but also the logical time at which a data item is manipulated.

**Overhead.** Works in the data analytics domain report space and time overhead of recording provenance (compared to an execution not collecting provenance). However, the evaluation is not based on a unified benchmark. Dataset sizes range from ten to hundred thousands of tuples [9], over a few Gigabytes [153] to hundreds of Gigabytes [3, 104, 106]. The runtime overhead reported across the solutions ranges from about 10% [3, 106, 122] in good cases to more than 100% in unfavorable test scenarios [9, 122]. To the best of our knowledge, no formal complexity analysis on collecting provenance in data analytics systems has been conducted.

### *4.3.4 General programming*

Provenance solutions for general purpose programming languages like C++, Java, or Python have become subject to research in the recent years. Unlike, for instance, SWfMSs, the languages do not come with inherent provenance support. As a consequence, multiple different approaches to collect provenance are under research.

**Workflow characteristics.** Workflow graphs $W$ derived from programs written in general programming languages may contain edges $P$ representing data dependencies (i.e., changes of variable values) as well as control dependencies (i.e. control loops or conditions). A consequence of the mixture also is the presence of cycles in $W$.

**Granularity.** Both fine-grained and coarse-grained provenance solutions for programming languages exist. For instance, noWorkflow collects fine-grained provenance for individual variables or items in collections [143]. In contrast, YesWorkflow provides coarse-grained provenance by limiting provenance information to the level of function boundaries [127]. We observe that in this domain, several solutions refer to fine-grained provenance. While these solutions indeed capture more details than those they compare to (and are thus finer-grained), they do not qualify as fine-grained according to our definition.

**Form.** In the context of programming languages, all forms of provenance are under research. For instance, the provenance solutions Starflow [15] and NoWorkflow [143] aim at collecting fine-grained, retrospective provenance for each individual global and local variable. While Starflow and Jif require modifications or explicit modelling during the program definition, noWorkflow handles unmodified programs. In contrast to the named solutions, YesWorkflow generates prospective provenance [127] from annotations. It extracts building blocks and data dependencies solely from annotations made in a dedicated annotation language. Relying not only on a static analysis but also on annotations is common to collect prospective provenance in this domain [69, 127]. Efforts to combine retrospective and prospective provenance in the domain of general programming are also advancing [69, 159]. Recent research also puts emphasis on collecting evolution provenance for programs. Pimentel et al. [160] propose a solution based on noWorkflow that captures modifications to the program definition along with their execution provenance. For each program execution, input data, results, and the current version of the program are recorded.

**Utilized languages.** Many provenance solutions work across different languages [127, 143, 175]. NoWorkflow [143, 158], YesWorkflow [127], RDataTracker [120], and Starflow [15] aim at interpreted languages like Python, R, or Matlab. Provenance extensions also exist for compiled languages like C or C++. Tariq et al. introduce SPADE, a provenance solution capable of tracking and analyzing provenance from multiple possibly distributed sources including sources compiled with an extended LLVM compiler. It adds provenance instrumentation at compile time [79, 175]. In order to provide security guarantees on the data being processed the Jif programming language extends Java to internally capture the information flow [144]. Even though work on information flow security considerably resides in the context of provenance, a more detailed discussion on it is out of scope and can be found in [165].

**Provenance capture.** While many solutions have in common that they are applicable to multiple languages, their approach to track provenance differs. The first group of provenance solutions relies on program annotation or instrumentation. For instance, YesWorkflow obtains its provenance graph from annotations associated with each function definition [127]. Similarly, RDataTracker requires provenance library calls for each method in order to build a provenance graph [120]. The Jif programming language explicitly taints all data processed in a program [144]. In contrast to this group, the second one does not require any modification to the script. NoWorkflow belongs to this group [143]. It employs program slicing to identify data dependencies [158]. A third group of solutions makes use of both explicit provenance modifications to the program as well as transparent provenance collection. For instance, Starflow can track provenance without annotation. However, expressiveness and understandability of the provenance improves in case annotations are available [15]. Dey et al. and Pimentel et al. combine YesWorkflow and NoWorkflow to combine the more abstract, simpler to understand YesWorkflow provenance with the more detailed, less comprehensible NoWorkflow provenance in order to obtain well-explorable provenance with a high degree of details [69, 159].

**Overhead.** Evaluations provided in works in this domain focus on a qualitative assessment of the provenance expressiveness. An exception is SPADE [175]., where they report a runtime overhead of about 5% to 14% for provenance collection. Further quantitative measurements and detailed complexity analyses are yet to appear.

## 5 Requirements

Collecting, querying, or analyzing provenance of different types to achieve the various applications summarized in Sec. 2 needs to be performed within given system and application requirements. The choice of what provenance solution is best suited depends on various requirements, such as runtime or interoperability. This section presents a classification of *system requirements* to be taken into account and discusses different methods considering and possibly proposing optimizations with respect to these requirements.

### 5.1 Requirement classification

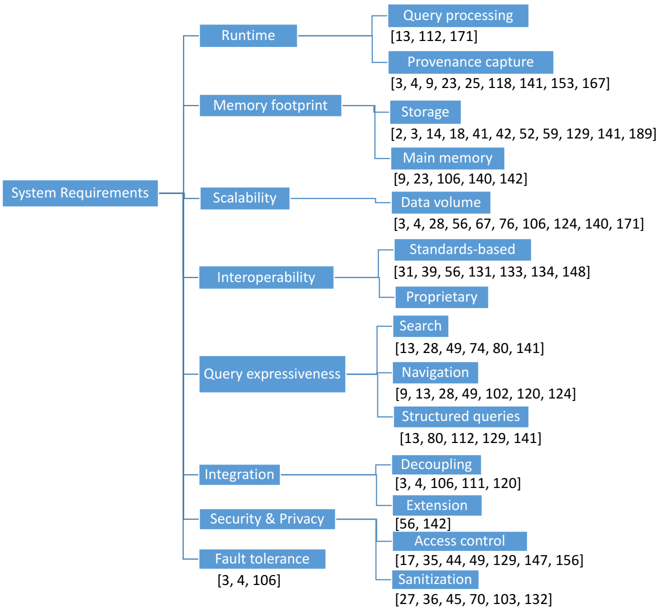

Fig. 6: Classification of requirements

Fig. 6 summarizes the proposed requirement classification. *Runtime* focuses on the time "lost" for leveraging provenance, and divides into cost of processing provenance queries and the cost for capturing provenance. Leveraging provenance typically also requires access and usage to both main memory and storage, leaving a *memory footprint*. *Scalability* requirements may apply, in particular on the data volume (of processed data and provenance). Another requirement is *interoperability* of provenance-enabled systems to easily exchange provenance, for which both standards-based or proprietary solutions apply. Concerning the *query expressiveness* of provenance queries, we distinguish between (selective) search queries, navigation queries, and structured queries. When it comes to system *integration* of the provenance capabilities, requirements vary from possibly tightly integrating provenance capabilities by extending the original program code to necessitating a decoupling of provenance management and the system for which provenance is used. As for any application handling data, *security and privacy* may be issues and systems may require different access control strategies and sanitization (or abstraction) techniques. Finally, *fault tolerance* plays a role especially in modern, highly distributed environments. We point out that there is a minor overlap of our proposed classification with a classification of provenance purposes [87]. The overlap limits to scalability, query expressiveness, and interoperability.

The following discussion provides examples of provenance systems where specific requirements have been explicitly considered and optimized for. The list of citations is not intended to be complete but to illustrate different aspects within the classes of the requirements classification. Note that Fig. 6 also summarizes the examples (by reference) provided in each class.

### 5.2 Runtime

We distinguish approaches optimizing runtime during provenance query processing or during provenance capture.
**Query processing.** As we have seen, different types of queries apply in provenance (further see Sec. 5.6). Consequently, different query optimization techniques apply. For instance, provenance browser [13], a visualization tool for provenance, uses optimization techniques proposed in [12] to improve user's query performance. Specialized indexing techniques for accelerated query processing have also been explored, e.g., in [112, 171].
**Provenance capture.** Many systems and algorithms, including [3, 4, 9, 23, 25, 118, 141, 153, 167] develop solutions to reduce provenance capture runtime overhead or at least experimentally study the runtime overheads. Optimizations are considered at various levels, from the way provenance is physically or logically modelled to algorithmic optimizations. Another aspect influencing the runtime of provenance capture is the choice of whether or not to capture provenance in an *eager* or *lazy* way [46]. Whereas eager provenance capture computes (and stores) provenance during processing, lazy provenance capture computes (selected) provenance when requested by a provenance query. Typically, eager provenance adds a higher one-time runtime overhead but improves query processing runtime, while lazy provenance capture has a priori less runtime overhead during processing at the price of a higher query processing runtime.

### 5.3 Memory footprint

Provenance solutions require memory resources, both on disk or main memory.
**Storage.** Concerning the storage on disk, optimizations reducing the required storage space of provenance either research adequate data models (e.g., [14, 18]) or opt for reducing the amount of provenance to be stored. The latter can be done either by restricting to an excerpt of the provenance [41, 52, 59, 129] based on application need or user-specification, or computing and storing provenance summaries (either exact, e.g., [14,42] or approximate [2]). These techniques typically incur some information loss that may however be acceptable for different types of applications. While not all provenance systems focus on optimizing required storage, several systems (e.g., [3, 141, 188]) provide experimental evidence to showcase the acceptable storage overhead incurred by provenance capture.
**Main memory.** Provenance-enabled systems primarily use main memory to perform caching of intermediate results. This typically aims for faster provenance capture [9, 23, 106, 140, 142]. However, we are not aware of any work studying or optimizing the load put on main memory when adding provenance capabilities to a system.

### 5.4 Scalability

Provenance systems must be capable to handle large amounts of provenance data that applications produce.

Ensuring the scalability to large volumes of data in a system's backend is possible by taking advantage of parallelization. For instance, Swift [76] includes a parallel scripting language and proposes a data model for recording and querying scientific workflow provenance. Swift supports scalability as it records provenance in large and distributed workflows. However, its storage management is not currently scalable and querying provenance can take a long time to execute. Another approach to enable scalability in the backend is to leverage and extend scalable systems (e.g., Cloud solutions or Map Reduce programs). This strategy is adopted by [3, 4, 56, 106, 140, 171].

As for scaling to large volumes of provenance data at the front-end, MapOrbiter [124], Inprov [28], or [67] include solutions to visualize and interact with large volumes of provenance. Both employ clustering of provenance entities to avoid the presentation of a cluttered provenance graph. However, clustering methods differ in the two works, indeed, MapOrbiter uses semantic information to group related nodes whereas InProv uses temporal clustering of information recorded in the same period.

### 5.5 Interoperability

Interoperability describes the ability of a provenance-enabled system to exchange provenance with other systems and to combine provenance produced by multiple systems.
**Standards-based.** To facilitate the exchange between provenance systems, standardization efforts have led to the definition of a W3C standard. A first standardization effort was the Open Provenance Model (OPM) [138], adopted in several workflow provenance systems such as Taverna [131, 134], Karma [39], Kepler-Hadoop [56] and COMAD-Kepler [31]. The W3C PROV [130] standard is deeply inspired by OPM. Several approaches further extend the standard [133, 148].
**Proprietary.** Despite the above standardization efforts, we observe that its adoption is not yet widely spread and that many tools, even though allowing for provenance export and import, rely on proprietary formats for exchanging provenance using a variety of data models, including semantic, relational, and semi-structured ones.

### 5.6 Query Expressiveness

Concerning the expressiveness of provenance queries, we distinguish between (selective) search queries, navigation queries, and structured queries.

**Search queries.** Several systems [13, 28, 74, 80, 141] allow to query the provenance graph through easy to use search queries such as searching by items (e.g., [49]), by time (e.g., [28]), or by kind of elements tracked (e.g., [13]).

**Navigation.** The extension of a dependency edge in a provenance graph and changing granularity (ZOOM IN/OUT) is for instance adopted in Map Orbiter [124], Provenance Curious [102], Provenance explorer [49], RDataTracker [120], Lipstick [9], and InProv [28]. We consider these as navigation queries through the provenance graph. Incremental queries provided by Provenance browser [13] also qualify as navigation queries since these allow to move, in the context of one perspective on provenance to the next.

**Structured query languages.** The most expressive from of queries are structured query languages. Since many provenance systems use relational, XML, or RDF storage, they support querying provenance by leveraging the corresponding query languages. Indeed, they customize existing query languages like XPath, SQL, or SPARQL in order to query the provenance store (examples include PERM [80], PQL [141], QLP [13], VQL [129], and ProQL [112]).

## 5.7 Application Integration

There are different means in extending an existing system with provenance capabilities. Provenance management can either be done at a layer decoupled from the main processing engine or a tight integration is implemented by extending the processing engine with provenance capabilities.

**Extension.** The tight integration implemented by extending the code of systems or the environment (e.g. the kernel) in which provenance should be tracked is a commonly adopted technique (e.g., [56, 72]). However, such tight integration is typically hard to implement for an already operative system and may not adapt well to new versions of the underlying (extended) software. Clearly, this imposes a high maintenance burden on the extended software.

**Decoupling.** Examples of systems capturing provenance by providing provenance-management decoupled from the actual processing include [3, 4, 106, 111, 120]. For instance, Titian [106] is built on the top of existing systems and integrates smoothly with the Spark runtime and programming interface. Another example is RDataTracker [120] that is implemented as a data provenance library for R scripts. As for [111], which uses provenance for collaborative data sharing, it is implemented as a layer above an RDBMS.

## 5.8 Security and Privacy

Making complete provenance available to all users may raise both security and privacy concerns [21], currently addressed either by implementing configurable access control mechanism or by implementing sanitization techniques.

**Access control.** Configurable access control organizes access to the resources provided by a provenance-supporting system with a granularity that is sufficient to protect these resources. For instance, [49] proposes an authentication system to redirect users following their access privileges to the suitable visualization of provenance. SecProv [44] is a visual interface that uses role-based access policy to determine which view of workflow provenance is accessible to specific users. Access control policies are specified during workflow design, and inherited by the derived provenance produced at execution time. Authors of [129] propose an approach to collect provenance for private Web services in the context of enterprises. Realizing that provenance must not be visible to all users, they suggest an access control mechanism for Web service runtimes including authentication and authorization features. Another security issue is the fact that as provenance capture is often implemented such that it executes with the same privileges and the same user space as the process itself, leading to new possible attacks. In this context, security of the system may be improved based on sandboxing the application [17]. Enhancing the security in the system is also the topic of [156]. To ease the definition of access control rules leveraging provenance, different languages have been proposed [35, 147].

**Sanitization.** Sanitization abstracts or prunes provenance by omitting sensitive pieces of the provenance. An overview of provenance-aware sanitization systems is provided in [47]. Cheney [45] proposes a framework for provenance abstraction that is based on formalization of security policies such as obfuscation and disclosure. Similarly, Cadenhead et al [36] present a graph rewriting approach to abstract away sensitive information within provenance. Zoom [27] applies sanitization for workflow provenance. Here, users, who are knowledgeable about the structure of the workflow, indicate which module they are interested in. A reduction of provenance is performed by grouping modules considered to be uninteresting. ProPub [70] computes provenance that could be published based on certain privacy requirements. It allows users to edit provenance by providing anonymizing, abstracting, and hiding operators. ProPub encompasses mechanisms to repair conflicts expected between user's manipulations and privacy requirements. Abstracting provenance conforming to W3C PROV given user credentials is studied in [132]. This work includes mechanisms to detect problems possibly occurring after abstraction operation, such as the appearance of cycles and the breaking of temporal constraints. Similarly [103] proposes graph rewriting approaches that omit sensitive provenance while preserving the integrity of the abstracted provenance graph.

As pointed out by [174], issues related to provenance security and privacy remain open issues. More specifically, it highlights the gap between current provenance frameworks used in distributed systems and requirements expected to ensure the security and privacy of provenance.

### 5.9 Fault tolerance

The last requirement of our classification to consider when devising a solution for provenance management is fault tolerance, i.e., the ability of the system to re-compute provenance when failure is detected. One solution is to use frameworks like Apache Hadoop or Apache Spark, as tasks for collecting provenance are distributed and replicated between nodes of the cluster just like any other task. Consequently, when a node fails to execute a sub-task of the provenance process, another one will take over the execution of this function. Current provenance systems taking advantage of these functionalities and thus implementing fault tolerance for provenance capture include [3, 4, 106].

## 6 Summary and Research Challenges

In the present survey, we have reviewed the state of the art of provenance research, focusing on the questions why capturing provenance is useful, what form captured provenance information may take for various types of processes, and which system requirements need to be considered by systems offering provenance capture.

The applications of provenance surveyed in this paper relate to understanding, reproducing, and improving the production processes of some derived data. Our classification further highlights the targeted users for both collecting and leveraging provenance for a given application.

Concerning the form provenance may take, we have presented a type hierarchy of provenance types that encompasses data provenance, workflow provenance, information system provenance, and the generic type provenance metadata. In the detailed discussion of data provenance, we have focused the discussion on novel techniques on the provenance of expected, but missing query results. The novel classification of systems collecting workflow provenance recognizes different domains of workflow provenance and distinguishes different provenance granularities and forms.

Our discussion of requirements for provenance computation reveals nine requirement classes such as efficiency, scalability, security, or query expressiveness. Considering and adequately coping with these requirements is essential to deploy and leverage provenance management systems for the various applications we have identified.

Achieving this overarching goal however requires to tacke several research challenges that so far have only been marginally addressed. We conclude this survey by four research opportunities to the provenance community that all aim at fostering the adoption of techniques capturing and exploiting provenance information that, despite the very relevant applications, is still quite modest.

**Challenge 1: Systems.** As pointed out in our discussion of requirements, systems that are more aware of and responsive to requirements of end-users still need to be developed. To this end, workload-based optimization of provenance collection is one possible avenue for future research. Here, the optimization requires to for instance automatically select an adequate type, model, granularity, compression, or suited collection points of provenance while satisfying possibly multiple user constraints.

**Challenge 2: Data provenance for further processes.** The most fine-grained type of provenance, namely data provenance, is so far achieved for a very restricted class of processes. While first efforts have studied to capture the same level of detail for workflow provenance, it remains an open question how provenance at the level of individual data items may be captured and rendered useful for other types of processes. For instance, in data cleaning or data integration processes, a typical problem is to find the right set of algorithms, parameters, or set of rules to forge the desired high-quality output. Enabling fine-grained provenance capture for these requires defining what provenance is relevant for the process (e.g.,for identified partial functional dependencies, it may be interesting to know which tuples violate the functional dependency, while for entity resolution rules, the information on the responsibility of each part of the rules for the duplicate classification is more interesting). This provenance then needs to be modelled, efficiently computed, managed, and queried In addition, provenance of missing results has so far not been considered beyond structured queries over relational data, offering plenty of research opportunities (both for further data models and processes).

**Challenge 3: Exploration and analysis of provenance data.** The abundance of provenance data that can be captured poses the new challenge of making use of these data. While querying provenance data has been studied together with data models for provenance, there exists only little work on properly visualizing, exploring, and analyzing provenance data in a user-friendly way. Interesting research questions to be answered are for instance how to adequately visualize different types of provenance for the various applications we have discussed, how to effectively support visual and interactive provenance data exploration, and how the abstraction and information loss caused by the visual encoding may impact on the provenance capture optimization and management, e.g., in terms of incremental computation or necessary granularity to be computed.

The challenges mentioned above are only some of the many questions that remain to be answered in the field of provenance. These challenges also have a high practical value so that overall, we believe that in the years to come, provenance research will remain an attractive research field both for foundational researchers and practitioners.

**Acknowledgements.** The authors thank the German Research Foundation (DFG) for financial support within project D03 of SFB/Transregio 161.

## References

1. U. Acar, P. Buneman, J. Cheney, J. Van Den Bussche, N. Kwasnikowska, and S. Vansummeren. A graph model of data and workflow provenance. In *Workshop on Theory and Practice of Provenance (TAPP)*, 2010.
2. E. Ainy, P. Bourhis, S. B. Davidson, D. Deutch, and T. Milo. Approximated Summarization of Data Provenance. In *Conference on Information and Knowledge Management (CIKM)*, pages 483–492, 2015.
3. S. Akoush, R. Sohan, and A. Hopper. HadoopProv: Towards Provenance As a First Class Citizen in MapReduce. In *Workshop on Theory and Practice of Provenance (TAPP)*, 2013.
4. A. Alkhaldi, I. Gupta, V. Raghavan, and M. Ghosh. Leveraging Metadata in No SQL Storage Systems. In *IEEE Conference on Cloud Computing (CLOUD)*, pages 57–64, 2015.
5. P. Alper, K. Belhajjame, C. A. Goble, and P. Karagoz. Enhancing and abstracting scientific workflow provenance for data publishing. In *EDBT/ICDT Workshops*, pages 313–318, 2013.
6. I. Altintas, O. Barney, and E. Jaeger-Frank. Provenance collection support in the Kepler scientific workflow system. In *International Provenance and Annotation Workshop (IPAW)*, pages 118–132, 2006.
7. P. Alvaro, J. Rosen, and J. M. Hellerstein. Lineage-driven Fault Injection. In *ACM Conference on the Management of Data (SIGMOD)*, pages 331–346, 2015.
8. B. Amann, C. Constantin, C. Caron, and P. Giroux. Weblab prov: Computing fine-grained provenance links for xml artifacts. In *EDBT/ICDT Workshops*, pages 298–306, 2013.
9. Y. Amsterdamer, S. B. Davidson, D. Deutch, T. Milo, J. Stoyanovich, and V. Tannen. Putting Lipstick on Pig : Enabling Database-style Workflow Provenance. *Proceedings of the VLDB Endowment (PVLDB)*, 5:346–357, 2011.
10. Y. Amsterdamer, D. Deutch, and V. Tannen. On the limitations of provenance for queries with difference. In *Workshop on Theory and Practice of Provenance (TAPP)*, 2011.
11. Y. Amsterdamer, D. Deutch, and V. Tannen. Provenance for aggregate queries. In *ACM Symposium on Principles of Database Systems (PODS)*, pages 153–164, 2011.
12. M. K. Anand, S. Bowers, and B. Ludäscher. Techniques for efficiently querying scientific workflow provenance graphs. In *Conference on Extending Database Technology (EDBT)*, pages 287–298, 2010.
13. M. K. Anand, S. Bowers, and B. Ludäscher. Provenance browser: Displaying and querying scientific workflow provenance graphs. In *IEEE International Conference on Data Engineering (ICDE)*, pages 1201–1204, 2010.
14. M. K. Anand, S. Bowers, T. McPhillips, and B. Ludäscher. Efficient provenance storage over nested data collections. In *Conference on Extending Database Technology (EDBT)*, pages 958–969, 2009.
15. E. Angelino, D. Yamins, and M. I. Seltzer. Starflow: A script-centric data analysis environment. In *International Provenance and Annotation Workshop (IPAW)*, pages 236–250, 2010.
16. B. S. Arab, D. Gawlick, V. Krishnaswamy, V. Radhakrishnan, and B. Glavic. Reenactment for read-committed snapshot isolation. In *Conference on Information and Knowledge Management (CIKM)*, pages 841–850, 2016.
17. N. Balakrishnan, T. Bytheway, L. Carata, R. Sohan, and A. Hopper. Towards secure user-space provenance capture. In *Workshop on Theory and Practice of Provenance (TAPP)*, 2016.
18. R. S. Barga and L. A. Digiampietri. Automatic capture and efficient storage of e-Science experiment provenance. *Concurrency Computation Practice and Experience*, 20(5):419–429, 2008.
19. C. Batini and M. Scannapieco. *Data Quality: Concepts, Methodologies and Techniques*. Springer-Verlag New York, Inc., 2006.
20. L. Bavoil, S. P. Callahan, P. J. Crossno, J. Freire, C. E. Scheidegger, C. T. Silva, and H. T. Vo. Vistrails: enabling interactive multiple-view visualizations. In *IEEE Visualization (VIS)*, pages 135–142, 2005.
21. E. Bertino, G. Ghinita, M. Kantarcioglu, D. Nguyen, J. Park, R. Sandhu, S. Sultana, B. Thuraisingham, and S. Xu. A roadmap for privacy-enhanced secure data provenance. *Journal of Intelligent Information Systems*, 43(3):481–501, 2014.
22. D. Bhagwat, L. Chiticariu, W. C. Tan, and G. Vijayvargiya. An annotation management system for relational databases. *The VLDB Journal*, 14(4), 2005.
23. N. Bidoit, M. Herschel, and A. Tzompanaki. Efficient computation of polynomial explanations of why-not questions. In *Conference on Information and Knowledge Management (CIKM)*, pages 713–722, 2015.
24. N. Bidoit, M. Herschel, and K. Tzompanaki. Immutably answering why-not questions for equivalent conjunctive queries. In *Workshop on Theory and Practice of Provenance (TAPP)*, 2014.
25. N. Bidoit, M. Herschel, and K. Tzompanaki. Query-based why-not provenance with NedExplain. In *Conference on Extending Database Technology (EDBT)*, pages 145–156, 2014.
26. N. Bidoit, M. Herschel, and K. Tzompanaki. EFQ: why-not answer polynomials in action. *Proceedings of the VLDB Endowment (PVLDB)*, 8(12):1980–1983, 2015.
27. O. Biton, S. Cohen-Boulakia, S. B. Davidson, and C. S. Hara. Querying and managing provenance through user views in scientific workflows. In *IEEE International Conference on Data Engineering (ICDE)*, pages 1072–1081, 2008.
28. M. A. Borkin, C. S. Yeh, M. Boyd, P. Macko, K. Z. Gajos, M. Seltzer, and H. Pfister. Evaluation of filesystem provenance visualization tools. *IEEE Transactions on Visualization and Computer Graphics*, 19(12):2476–2485, 2013.
29. S. Börzsönyi, D. Kossmann, and K. Stocker. The skyline operator. In *IEEE International Conference on Data Engineering (ICDE)*, pages 421–430, 2001.
30. P. Bourhis, D. Deutch, and Y. Moskovitch. POLYTICS: provenance-based analytics of data-centric applications. In *IEEE International Conference on Data Engineering (ICDE)*, pages 1373–1374, 2017.
31. S. Bowers, T. M. McPhillips, and B. Ludäscher. Provenance in collection-oriented scientific workflows. *Concurrency and Computation: Practice and Experience*, 20(5):519–529, 2008.
32. S. Bowers, T. M. McPhillips, S. Riddle, M. K. Anand, and B. Ludäscher. Kepler/pPOD: Scientific workflow and provenance support for assembling the tree of life. In *International Provenance and Annotation Workshop (IPAW)*, pages 70–77, 2008.
33. P. Buneman, S. Khanna, and W. C. Tan. Why and where: A characterization of data provenance. In *International Conference on Database Theory (ICDT)*, pages 316–330, 2001.
34. P. Buneman, S. Khanna, and W. C. Tan. On propagation of deletions and annotations through views. In *ACM Symposium on Principles of Database Systems (PODS)*, pages 150–158, 2002.
35. T. Cadenhead, V. Khadilkar, M. Kantarcioglu, and B. Thuraisingham. A language for provenance access control. In *ACM Conference on Data and Application Security and Privacy (CODASPY)*, pages 133–144, 2011.
36. T. Cadenhead, V. Khadilkar, M. Kantarcioglu, and B. Thuraisingham. Transforming Provenance Using Redaction. In *ACM*

*Symposium on Access Control Models and Technologies (SACMAT)*, pages 93–102, 2011.

37. S. P. Callahan, J. Freire, E. Santos, C. E. Scheidegger, T. Vo, and H. T. Silva. VisTrails : Visualization meets Data Management. *ACM Conference on the Management of Data (SIGMOD)*, pages 745–747, 2006.
38. D. Calvanese, M. Ortiz, M. Simkus, and G. Stefanoni. Reasoning about explanations for negative query answers in DL-Lite. *Journal on Artificial Intelligence Research (JAIR)*, 48:635–669, 2013.
39. B. Cao, B. Plale, G. Subramanian, E. Robertson, and Y. Simmhan. Provenance information model of Karma version 3. In *Congress on Services - I (SERVICES)*, pages 348–351, 2009.
40. Y. Cao, C. Jones, T. Mcphillips, M. B. Jones, B. Ludäscher, P. Missier, C. Schwalm, P. Slaughter, D. Vieglais, L. Walker, and Y. Wei. DataONE: A Data Federation with Provenance Support. In *International Provenance and Annotation Workshop (IPAW)*, pages 230–234, 2016.
41. C. Caron, B. Amann, C. Constantin, and P. Giroux. WePIGE: The Weblab provenance information generator and explorer. In *Conference on Extending Database Technology (EDBT)*, pages 664–667, 2014.
42. A. Chapman, H. Jagadish, and P. Ramanan. Efficient provenance storage. *ACM Conference on the Management of Data (SIGMOD)*, pages 993–1006, 2008.
43. A. Chapman and H. V. Jagadish. Why not? In *ACM Conference on the Management of Data (SIGMOD)*, pages 523–534, 2009.
44. A. Chebotko, S. Lu, S. Chang, F. Fotouhi, and P. Yang. Secure abstraction views for scientific workflow provenance querying. *IEEE Transactions on Services Computing*, 3(4):322–337, 2010.
45. J. Cheney. A formal framework for provenance security. In *IEEE Computer Security Foundations Symposium (CSF)*, pages 281–293, 2011.
46. J. Cheney, L. Chiticariu, and W. C. Tan. Provenance in databases: Why, how, and where. *Foundations and Trends in Databases*, 1(4), 2009.
47. J. Cheney and R. Perera. An analytical survey of provenance sanitization. In *International Provenance and Annotation Workshop (IPAW)*, pages 113–126, 2014.
48. S. Chester and I. Assent. Explanations for skyline query results. In *Conference on Extending Database Technology (EDBT)*, pages 349–360, 2015.
49. K. Cheung and J. Hunter. Provenance explorer - customized provenance views using semantic inferencing. In *International Semantic Web Conference (ISWC)*, pages 215–227, 2006.
50. F. Chirigati, D. Shasha, and J. Freire. ReproZip: Using Provenance to Support Computational Reproducibility. In *Workshop on Theory and Practice of Provenance (TAPP)*, pages 1–4, 2013.
51. L. Chiticariu and W. C. Tan. Debugging schema mappings with routes. In *Conference on Very Large Data Bases (VLDB)*, pages 79–90, 2006.
52. Z. Chothia, J. Liagouris, F. McSherry, and T. Roscoe. Explaining outputs in modern data analytics. *Proceedings of the VLDB Endowment (PVLDB)*, 9(12):1137–1148, 2016.
53. E. Commission. *Horse meat: one year after - Actions announced and delivered!*, 2014 (accessed March 15, 2016).
54. K. Cranmer, L. Heinrich, R. Jones, and D. M. South. Analysis preservation in ATLAS. *Journal of Physics*, 664, 2015.
55. D. Crawl and I. Altintas. A provenance-based fault tolerance mechanism for scientific workflows. In *International Provenance and Annotation Workshop (IPAW)*, pages 152–159, 2008.
56. D. Crawl, J. Wang, and I. Altintas. Provenance for mapreduce-based data-intensive workflows. In *Workshop on Workflows in Support of Large-Scale Science (WORKS)*, pages 21–30, 2011.
57. Y. Cui and J. Widom. Lineage tracing for general data warehouse transformations. In *Conference on Very Large Data Bases (VLDB)*, pages 471–480, 2001.
58. Y. Cui, J. Widom, and J. L. Wiener. Tracing the lineage of view data in a warehousing environment. *ACM Transactions on Database Systems (TODS)*, 25(2):179 – 227, 2000.
59. F. Curbera, Y. N. Doganata, A. Martens, N. Mukhi, and A. Slominski. Business provenance - A technology to increase traceability of end-to-end operations. In *On the Move to Meaningful Internet Systems OTM*, pages 100–119, 2008.
60. C. Dai, D. Lin, E. Bertino, and M. Kantarcioglu. An approach to evaluate data trustworthiness based on data provenance. In *Workshop on Secure Data Management (SDM)*, pages 82–98, 2008.
61. S. B. Davidson, S. Cohen-Boulakia, A. Eyal, B. Ludäscher, T. M. McPhillips, S. Bowers, M. K. Anand, and J. Freire. Provenance in scientific workflow systems. *IEEE Data Engineering Bulletin*, 30(4):44–50, 2007.
62. S. B. Davidson and J. Freire. Provenance and scientific workflows: Challenges and opportunities. In *ACM Conference on the Management of Data (SIGMOD)*, pages 1345–1350, 2008.
63. T. De Nies, I. Taxidou, A. Dimou, R. Verborgh, P. M. Fischer, E. Mannens, and R. de Walle. Towards Multi-level Provenance Reconstruction of Information Diffusion on Social Media. *Conference on Information and Knowledge Management (CIKM)*, pages 1823–1826, 2015.
64. E. Deelman, G. B. Berriman, A. L. Chervenak, Ó. Corcho, P. T. Groth, and L. Moreau. Metadata and provenance management. In A. Shoshani and D. Rotem, editors, *Scientific Data Management: Challenges, Technology, and Deployment*. Chapman & Hall/CRC, 2009.
65. E. Deelman, G. Singh, M. Su, J. Blythe, Y. Gil, C. Kesselman, G. Mehta, K. Vahi, G. B. Berriman, J. Good, A. C. Laity, J. C. Jacob, and D. S. Katz. Pegasus: A framework for mapping complex scientific workflows onto distributed systems. *Scientific Programming*, 13(3):219–237, 2005.
66. E. Dellis and B. Seeger. Efficient computation of reverse skyline queries. In *Conference on Very Large Data Bases (VLDB)*, pages 291–302, 2007.
67. D. Deutch, A. Gilad, and Y. Moskovitch. selP: Selective tracking and presentation of data provenance. In *IEEE International Conference on Data Engineering (ICDE)*, pages 1484–1487, 2015.
68. D. Deutch, Y. Moskovitch, and V. Tannen. A provenance framework for data-dependent process analysis. *Proceedings of the VLDB Endowment*, 7(6):457–468, feb 2014.
69. S. Dey, K. Belhajjame, D. Koop, M. Raul, and B. Ludäscher. Linking prospective and retrospective provenance in scripts. In *Workshop on Theory and Practice of Provenance (TAPP)*, 2015.
70. S. C. Dey, D. Zinn, and B. Ludäscher. Propub: Towards a declarative approach for publishing customized, policy-aware provenance. In *Conference on Scientific and Statistical Database Management (SSDBM)*, pages 225–243, 2011.
71. T. Ellkvist, D. Koop, E. W. Anderson, J. Freire, and C. T. Silva. Using provenance to support real-time collaborative design of workflows. In *International Provenance and Annotation Workshop (IPAW)*, pages 266–279, 2008.
72. S. Fehrenbach and J. Cheney. Language-integrated provenance. In *Symposium on Principles and Practice of Declarative Programming (PPDP)*, pages 214–227, 2016.
73. J. N. Foster, T. J. Green, and V. Tannen. Annotated XML: queries and provenance. In *ACM Symposium on Principles of Database Systems (PODS)*, pages 271–280, 2008.
74. J. Freire, D. Koop, E. Santos, and C. T. Silva. Provenance for computational tasks: A survey. *Computing in Science and Engineering*, 10(3):11–21, 2008.
75. J. Freire, C. T. Silva, S. P. Callahan, E. Santos, C. E. Scheidegger, and H. T. Vo. Managing rapidly-evolving scientific workflows. In *International Provenance and Annotation Workshop (IPAW)*, pages 10–18, 2006.
76. L. M. R. Gadelha, B. Clifford, M. Mattoso, M. Wilde, and I. Foster. Provenance management in Swift. *Future Generation Computer Systems*, 27(6):775–780, 2011.

77. Y. Gao, Q. Liu, G. Chen, B. Zheng, and L. Zhou. Answering why-not questions on reverse top-k queries. *Proceedings of the VLDB Endowment (PVLDB)*, 8(7):738–749, 2015.
78. D. Garijo, Ó. Corcho, and Y. Gil. Detecting common scientific workflow fragments using templates and execution provenance. In *International Conference on Knowledge Capture (K-CAP)*, pages 33–40, 2013.
79. A. Gehani and D. Tariq. SPADE: Support for Provenance Auditing in Distributed Environments. In *Proceedings of the International Middleware Conference*, pages 101–120, 2012.
80. B. Glavic and G. Alonso. The perm provenance management system in action. In *ACM Conference on the Management of Data (SIGMOD)*, pages 1055–1058, 2009.
81. B. Glavic, G. Alonso, R. J. Miller, and L. M. Haas. TRAMP: understanding the behavior of schema mappings through provenance. *Proceedings of the VLDB Endowment (PVLDB)*, 3(1):1314–1325, 2010.
82. B. Glavic, K. S. Esmaili, P. M. Fischer, and N. Tatbul. Ariadne: managing fine-grained provenance on data streams. In *Conference on Distributed Event-Based Systems (DEBS)*, pages 39–50, 2013.
83. C. Goble. Position Statement: Musings on Provenance, Workflow and (Semantic Web) Annotations for Bioinformatics. In *Workshop on Data Derivation and Provenance*, pages 152–159, 2002.
84. J. Goecks, A. Nekrutenko, and J. Taylor. Galaxy: a comprehensive approach for supporting accessible, reproducible, and transparent computational research in the life sciences. *Genome biology*, 11(8):R86, 2010.
85. T. J. Green, G. Karvounarakis, and V. Tannen. Provenance semirings. In *ACM Symposium on Principles of Database Systems (PODS)*, pages 31–40, 2007.
86. T. J. Green, G. Karvounarakis, N. E. Taylor, O. Biton, Z. G. Ives, and V. Tannen. ORCHESTRA: facilitating collaborative data sharing. In *ACM Conference on the Management of Data (SIGMOD)*, pages 1131–1133, 2007.
87. P. Groth, Y. Gil, J. Cheney, and S. Miles. Requirements for Provenance on the Web. *International Journal of Digital Curation*, 7(1):39–56, 2012.
88. P. Groth, S. Miles, W. Fang, S. C. Wong, K.-P. Zauner, and L. Moreau. Recording and using provenance in a protein compressibility experiment. In *IEEE Symposium on High Performance Distributed Computing (HPDC)*, pages 201–208, 2005.
89. P. Groth and L. Moreau. *PROV-Overview: An Overview of the PROV Family of Documents*, 2013 (accessed March 15, 2016).
90. Grust and Rittinger. Observing sql queries in their natural habitat. *ACM Transactions on Database Systems (TODS)*, 2012.
91. O. Hartig and J. Zhao. Using web data provenance for quality assessment. In *Workshop on the Role of Semantic Web in Provenance Management (SWPM)*, 2009.
92. Z. He and E. Lo. Answering why-not questions on top-k queries. In *IEEE International Conference on Data Engineering (ICDE)*, pages 750–761, 2012.
93. Z. He and E. Lo. Answering why-not questions on top-k queries. *IEEE Transactions on Knowledge and Data Engineering (TKDE)*, 26(6):1300–1315, 2014.
94. M. Herschel. A hybrid approach to answering why-not questions on relational query results. *ACM Journal on Data and Information Quality (JDIQ)*, 5(3):10:1–10:29, 2015.
95. M. Herschel and H. Eichelberger. The Nautilus Analyzer: understanding and debugging data transformations. In *Conference on Information and Knowledge Management (CIKM)*, pages 2731–2733, 2012.
96. M. Herschel and T. Grust. Transformation lifecycle management with Nautilus. In *Workshop on the Quality of Data (QDB)*, 2011.
97. M. Herschel and M. A. Hernández. Explaining missing answers to SPJUA queries. *Proceedings of the VLDB Endowment (PVLDB)*, 3(1):185–196, 2010.
98. M. Herschel and M. Hlawatsch. Provenance: On and behind the screens. In *ACM Conference on the Management of Data (SIGMOD)*, pages 2213–2217, 2016.
99. M. Hlawatsch, M. Burch, F. Beck, J. Freire, C. Silva, and D. Weiskopf. Visualizing the evolution of module workflows. In *International Conference on Information Visualisation (IV)*, pages 40–49, 2015.
100. R. Hoekstra and P. Groth. Prov-o-viz - understanding the role of activities in provenance. In *International Provenance and Annotation Workshop (IPAW)*, pages 215–220, 2014.
101. J. Huang, T. Chen, A. Doan, and J. F. Naughton. On the provenance of non-answers to queries over extracted data. *Proceedings of the VLDB Endowment (PVLDB)*, 1(1):736–747, 2008.
102. M. R. Huq, P. M. G. Apers, and A. Wombacher. Provenance-curious: a tool to infer data provenance from scripts. In *Conference on Extending Database Technology (EDBT)*, pages 765–768, 2013.
103. J. Hussein, L. Moreau, and V. Sassone. Obscuring provenance confidential information via graph transformation. In *Conference on Trust Management (IFIP)*, pages 109–125, 2015.
104. R. Ikeda, H. Park, and J. Widom. Provenance for Generalized Map and Reduce Workflows. In *Conference on Innovative Data Systems Research (CIDR)*, pages 273–283, 2011.
105. T. Imieliński and W. Lipski, Jr. Incomplete information in relational databases. *Journal of the ACM*, 31(4):761–791, 1984.
106. M. Interlandi, K. Shah, S. D. Tetali, M. A. Gulzar, S. Yoo, M. Kim, T. Millstein, and T. Condie. Titian: data provenance support in Spark. *Proceedings of the VLDB Endowment (PVLDB)*, 9(3):216–227, 2015.
107. M. S. Islam, C. Liu, and R. Zhou. Flexiq: A flexible interactive querying framework by exploiting the skyline operator. *Journal of Systems and Software*, 97:97–117, 2014.
108. M. S. Islam, R. Zhou, and C. Liu. On answering why-not questions in reverse skyline queries. In *IEEE International Conference on Data Engineering (ICDE)*, pages 973–984, 2013.
109. L. Karsai, A. Fekete, J. Kay, and P. Missier. Clustering provenance facilitating provenance exploration through data abstraction. In *Workshop on Human-In-the-Loop Data Analytics (HILDA)*, pages 6:1–6:5, 2016.
110. G. Karvounarakis and T. J. Green. Semiring-annotated data: Queries and provenance? *SIGMOD Record*, 41(3):5–14, 2012.
111. G. Karvounarakis, T. J. Green, Z. G. Ives, and V. Tannen. Collaborative data sharing via update exchange and provenance. *ACM Transactions on Database Systems (TODS)*, 38(3):19:1–19:42, 2013.
112. G. Karvounarakis, Z. G. Ives, and V. Tannen. Querying data provenance. In *ACM Conference on the Management of Data (SIGMOD)*, pages 951–962, 2010.
113. R. K. L. Ko and M. A. Will. Progger: An efficient, tamper-evident kernel-space logger for cloud data provenance tracking. In *IEEE Conference on Cloud Computing (CLOUD)*, pages 881–889, 2014.
114. S. Köhler, B. Ludäscher, and D. Zinn. First-order provenance games. In *In Search of Elegance in the Theory and Practice of Computation*, pages 382–399, 2013.
115. S. Köhler, S. Riddle, D. Zinn, T. M. McPhillips, and B. Ludäscher. Improving workflow fault tolerance through provenance-based recovery. In *Conference on Scientific and Statistical Database Management (SSDBM)*, pages 207–224, 2011.
116. V. Korolev and A. Joshi. PROB: A tool for Tracking Provenance and Reproducibility of Big Data Experiments. In *Reproduce, HPCA*, pages 264–286, 2014.
117. S. Krishnan, J. Wang, M. J. Franklin, K. Goldberg, and T. Kraska. Privateclean: Data cleaning and differential privacy. In *ACM Conference on the Management of Data (SIGMOD)*, pages 937–951, 2016.

118. D. Kulkarni. A Provenance Model for Key-value Systems. *Workshop on Theory and Practice of Provenance (TAPP)*, pages 12:1–12:4, 2013.
119. N. Kwasnikowska and J. Van den Bussche. Mapping the NRC Dataflow Model to the Open Provenance Model. In *Workshop on Theory and Practice of Provenance (TAPP)*, pages 3–16, 2008.
120. B. Lerner and E. R. Boose. RDataTracker: Collecting provenance in an interactive scripting environment. In *Workshop on Theory and Practice of Provenance (TAPP)*, pages 1–4, 2014.
121. H. R. Lipford, F. Stukes, W. Dou, M. E. Hawkins, and R. Chang. Helping users recall their reasoning process. In *IEEE Conference on Visual Analytics Science and Technology (VAST)*, pages 187–194, 2010.
122. D. Logothetis, S. De, and K. Yocum. Scalable lineage capture for debugging DISC analytics. In *Symposium on Cloud Computing (SOCC)*, pages 1–15, 2013.
123. P. Macko and M. Chiarini. Collecting provenance via the xen hypervisor. In *Workshop on Theory and Practice of Provenance (TAPP)*, 2011.
124. P. Macko and M. Seltzer. Provenance map orbiter: Interactive exploration of large provenance graphs. In *Workshop on Theory and Practice of Provenance (TAPP)*, 2011.
125. A. Martens, A. Slominski, G. T. Lakshmanan, and N. Mukhi. Advanced case management enabled by business provenance. In *International Conference on Web Services (ICWS)*, pages 639–641, 2012.
126. T. McPhillips, S. Bowers, D. Zinn, and B. Ludäscher. Scientific workflow design for mere mortals. *Future Generation Computer Systems*, 25(5):541–551, 2009.
127. T. M. McPhillips, T. Song, T. Kolisnik, S. Aulenbach, K. Belhajjame, K. Bocinsky, Y. Cao, F. Chirigati, S. C. Dey, J. Freire, D. N. Huntzinger, C. Jones, D. Koop, P. Missier, M. Schildhauer, C. R. Schwalm, Y. Wei, J. Cheney, M. Bieda, and B. Ludäscher. YesWorkflow: A user-oriented, language-independent tool for recovering workflow information from scripts. *International Journal of Digital Curation*, 10(1):298–313, 2015.
128. A. Meliou, W. Gatterbauer, K. F. Moore, and D. Suciu. The complexity of causality and responsibility for query answers and non-answers. *Proceedings of the VLDB Endowment (PVLDB)*, 4(1):34 – 45, 2010.
129. A. Michlmayr, F. Rosenberg, P. Leitner, and S. Dustdar. Service provenance in qos-aware web service runtimes. In *International Conference on Web Services (ICWS)*, pages 115–122, 2009.
130. P. Missier, K. Belhajjame, and J. Cheney. The W3C PROV family of specifications for modelling provenance metadata. In *Conference on Extending Database Technology (EDBT)*, pages 773–776, 2013.
131. P. Missier, K. Belhajjame, J. Zhao, M. Roos, and C. A. Goble. Data lineage model for taverna workflows with lightweight annotation requirements. In *International Provenance and Annotation Workshop (IPAW)*, pages 17–30, 2008.
132. P. Missier, J. Bryans, C. Gamble, V. Curcin, and R. Danger. ProvAbs: Model, Policy, and Tooling for Abstracting PROV Graphs. In *International Provenance and Annotation Workshop (IPAW)*, pages 3–15, 2014.
133. P. Missier, S. Dey, K. Belhajjame, V. Cuevas-Vicenttín, and B. Ludäscher. D-prov: Extending the prov provenance model with workflow structure. In *Workshop on Theory and Practice of Provenance (TAPP)*, pages 9:1–9:7, 2013.
134. P. Missier and C. Goble. Workflows to open provenance graphs, round-trip. *Future Generation Computer Systems*, 27(6):812–819, 2011.
135. P. Missier, N. W. Paton, and K. Belhajjame. Fine-grained and efficient lineage querying of collection-based workflow provenance. In *Conference on Extending Database Technology (EDBT)*, pages 299–310, 2010.
136. L. Moreau. The foundations for provenance on the web. *Foundations and Trends in Web Science*, 2(2-3):99–241, 2010.
137. L. Moreau. Provenance-based reproducibility in the semantic web. *Journal of Web Semantics*, 9(2):202–221, 2011.
138. L. Moreau, J. Freire, J. Futrelle, R. McGrath, J. Myers, and P. Paulson. The Open Provenance Model. *Future Generation Computer Systems*, 27(6):743–756, 2011.
139. T. Müller and T. Grust. Provenance for SQL through abstract interpretation: Value-less, but worthwhile. *Proceedings of the VLDB Endowment (PVLDB)*, 8(12):1872–1875, 2015.
140. K. Muniswamy-Reddy, P. Macko, and M. I. Seltzer. Provenance for the cloud. In *USENIX Conference on File and Storage Technologies (FAST)*, pages 197–210, 2010.
141. K.-K. Muniswamy-Reddy, U. Braun, D. A. Holland, P. Macko, D. Maclean, D. Margo, M. Seltzer, and R. Smogor. Layering in provenance systems. In *USENIX Annual Technical Conference*, 2009.
142. K.-K. Muniswamy-Reddy, D. A. Holland, U. Braun, and M. Seltzer. Provenance-aware storage systems. In *USENIX Annual Technical Conference*, pages 43–56, 2006.
143. L. Murta, V. Braganholo, F. Chirigati, D. Koop, and J. Freire. noWorkflow: Capturing and analyzing provenance of scripts. In *International Provenance and Annotation Workshop (IPAW)*, pages 71–83, 2014.
144. A. C. Myers. JFlow: Practical Mostly-Static Information Flow Control. In *Proceedings of the Symposium on Principles of Programming Languages (POPL)*, number January, pages 228–241, 1999.
145. M. Nagappan and M. a. Vouk. A Model for Sharing of Confidential Provenance Information in a Query Based System. *International Provenance and Annotation Workshop (IPAW)*, pages 62–69, 2008.
146. S. New. The transparent supply chain. *Harvard Business Review*, 2010.
147. Q. Ni, S. Xu, E. Bertino, R. Sandhu, and W. Han. An access control language for a general provenance model. In *Workshop on Secure Data Management (SDM)*, pages 68–88, 2009.
148. T. D. Nies, S. Coppens, R. Verborgh, M. V. Sande, E. Mannens, R. V. D. Walle, D. Nies, V. Sande, V. D. Walle, L. E. Access, and S. Towards. Easy Access to Provenance: an Essential Step Towards Trust on the Web. In *Computer Software and Applications Conference Workshops (COMPSACW)*, 2013.
149. X. Niu, R. Kapoor, B. Glavic, D. Gawlick, Z. H. Liu, V. Krishnaswamy, and V. Radhakrishnan. Interoperability for provenance-aware databases using PROV and JSON. In *Workshop on Theory and Practice of Provenance (TAPP)*, 2015.
150. T. M. Oinn, M. Addis, J. Ferris, D. Marvin, M. Senger, R. M. Greenwood, T. Carver, K. Glover, M. R. Pocock, A. Wipat, and P. Li. Taverna: a tool for the composition and enactment of bioinformatics workflows. *Bioinformatics*, 20(17):3045–3054, 2004.
151. T. M. Oinn, R. M. Greenwood, M. Addis, M. N. Alpdemir, J. Ferris, K. Glover, C. A. Goble, A. Goderis, D. Hull, D. Marvin, P. Li, P. W. Lord, M. R. Pocock, M. Senger, R. Stevens, A. Wipat, and C. Wroe. Taverna: lessons in creating a workflow environment for the life sciences. *Concurrency and Computation: Practice and Experience*, 18(10):1067–1100, 2006.
152. W. Oliveira, P. Missier, K. Ocaña, D. de Oliveira, and V. Braganholo. Analyzing Provenance Across Heterogeneous Provenance Graphs. In *International Provenance and Annotation Workshop (IPAW)*, pages 57–70, 2016.
153. C. Olston and B. Reed. Inspector gadget: A framework for custom monitoring and debugging of distributed dataflows. *Proceedings of the VLDB Endowment (PVLDB)*, 4(12):1237–1248, 2011.
154. C. Olston, B. Reed, U. Srivastava, R. Kumar, and A. Tomkins. Pig Latin: A Not-So-Foreign Language for Data Processing. In *ACM Conference on the Management of Data (SIGMOD)*, pages 1099–1110, 2008.

155. D. Papadias, Y. Tao, G. Fu, and B. Seeger. An optimal and progressive algorithm for skyline queries. In *ACM Conference on the Management of Data (SIGMOD)*, pages 467–478, 2003.
156. J. Park, D. Nguyen, and R. Sandhu. A provenance-based access control model. In *Conference on Privacy, Security and Trust (PST)*, pages 137–144, 2012.
157. Q. Pham, T. Malik, and I. Foster. Using provenance for repeatability. In *Workshop on Theory and Practice of Provenance (TAPP)*, 2013.
158. J. a. F. Pimentel, J. Freire, L. Murta, and V. Braganholo. Fine-Grained Provenance Collection over Scripts Through Program Slicing. In *International Provenance and Annotation Workshop (IPAW)*, pages 199–203, 2016.
159. J. F. Pimentel, S. Dey, T. McPhillips, K. Belhajjame, D. Koop, L. Murta, V. Braganholo, and B. Ludäscher. Yin & Yang: Demonstrating Complementary Provenance from noWorkflow & YesWorkflow. In *International Provenance and Annotation Workshop (IPAW)*, pages 161–165, 2016.
160. J. F. Pimentel, J. Freire, V. Braganholo, and L. Murta. Tracking and analyzing the evolution of provenance from scripts. In *International Provenance and Annotation Workshop (IPAW)*, pages 16–28, 2016.
161. A. Prabhune, A. Zweig, R. Stotzka, M. Gertz, and J. Hesser. Prov2ONE: An Algorithm for Automatically Constructing ProvONE Provenance Graphs. In *International Provenance and Annotation Workshop (IPAW)*, pages 204–208, 2016.
162. E. D. Ragan, A. Endert, J. Sanyal, and J. Chen. Characterizing provenance in visualization and data analysis: An organizational framework of provenance types and purposes. In *IEEE Transactions on Visualization and Computer Graphics*, pages 31 – 40, 2015.
163. S. Riddle, S. Köhler, and B. Ludäscher. Towards constraint provenance games. In *Workshop on Theory and Practice of Provenance (TAPP)*, 2014.
164. S. Roy, L. Chiticariu, V. Feldman, F. Reiss, and H. Zhu. Provenance-based dictionary refinement in information extraction. In *ACM Conference on the Management of Data (SIGMOD)*, pages 457–468, 2013.
165. A. Sabelfeld and A. C. Myers. Language-based information-flow security. *IIEEE Journal on Selected Areas in Communications*, 21(1):5–19, 2006.
166. Y. Simmhan, B. Plale, and D. Gannon. A survey of data provenance in e-science. *SIGMOD Record*, 34(3):31–36, 2005.
167. Y. L. Simmhan, B. Plale, and D. Gannon. Karma2: Provenance Management for Data Driven Workflows. *International Journal of Web Services Research*, 5(10):1–23, 2008.
168. Souilah, Francalanza, and Sassone. A formal model of provenance in distributed systems. In *Workshop on Theory and Practice of Provenance (TAPP)*, 2009.
169. H. Stitz, S. Luger, M. Streit, and N. Gehlenborg. AVOCADO: Visualization of Workflow-Derived Data Provenance for Reproducible Biomedical Research. In *European Conference on Visualization (EuroVis)*, pages 481–490, 2016.
170. C. H. Suen, R. K. L. Ko, Y. S. Tan, P. Jagadpramana, and B. Lee. S2logger: End-to-end data tracking mechanism for cloud data provenance. In *IEEE International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom)*, pages 594–602, 2013.
171. R. Szablocs, S. Aleksander, and D. Yurdaer. Large-Scale Distributed Storage Systems for Business Provenance. *IBM Research Report*, RC25154, 2011.
172. W. Tan, P. Missier, I. Foster, R. Madduri, D. De Roure, and C. Goble. A comparison of using Taverna and BPEL in building scientific workflows: The case of caGrid. *Concurrency Computation Practice and Experience*, 22(9):1098–1117, 2010.
173. W. C. Tan. Provenance in databases: Past, current, and future. *IEEE Data Engineering Bulletin*, 30(4):3–12, 2007.
174. Y. S. Tan, R. K. L. Ko, and G. Holmes. Security and data accountability in distributed systems: A provenance survey. In *IEEE Conference on High Performance Computing and Communications (HPCC)*, 2013.
175. D. Tariq, M. Ali, and A. Gehani. Towards automated collection of application-level data provenance. In *Workshop on Theory and Practice of Provenance (TAPP)*, 2012.
176. B. ten Cate, C. Civili, E. Sherkhonov, and W.-C. Tan. High-level why-not explanations using ontologies. In *ACM Symposium on Principles of Database Systems (PODS)*, pages 31–43, 2015.
177. Y. Theoharis, I. Fundulaki, G. Karvounarakis, and V. Christophides. On provenance of queries on semantic web data. *IEEE Internet Computing*, 15(1):31–39, 2011.
178. A. Thusoo, J. S. Sarma, N. Jain, Z. Shao, P. Chakka, S. Anthony, H. Liu, P. Wyckoff, and R. Murthy. Hive: A warehousing solution over a map-reduce framework. *Proceedings of the VLDB Endowment (PVLDB)*, 2(2):1626–1629, 2009.
179. Q. T. Tran and C.-Y. Chan. How to ConQueR why-not questions. In *ACM Conference on the Management of Data (SIGMOD)*, pages 15 – 26, 2010.
180. Q. T. Tran, C.-Y. Chan, and S. Parthasarathy. Query reverse engineering. *The VLDB Journal*, 23(5):721–746, 2014.
181. Tylissanakis and Cotroni. Data provenance and reproducibility in grid based scientific workflows. In *IEEE Workshop on Grid and Pervasive Computing Conference*, pages 42–49, 2009.
182. R. Y. Wang and D. M. Strong. Beyond accuracy: What data quality means to data consumers. *Journal of Management Information Systems*, 12(4):5–33, 1996.
183. Y. R. Wang, S. E. Madnick, et al. A polygen model for heterogeneous database systems: The source tagging perspective. In *Conference on Very Large Data Bases (VLDB)*, pages 519–538, 1990.
184. T. White. *Hadoop: The Definitive Guide*. O'Reilly Media, 4th edition, 2015.
185. A. Woodruff and M. Stonebraker. Supporting fine-grained data lineage in a database visualization environment. In *IEEE International Conference on Data Engineering (ICDE)*, pages 91–102, 1997.
186. M. Wylot, P. Cudré-Mauroux, and P. T. Groth. Tripleprov: efficient processing of lineage queries in a native RDF store. In *World Wide Web Conference (WWW)*, pages 455–466, 2014.
187. M. Zaharia, M. Chowdhury, M. J. Franklin, S. Shenker, and I. Stoica. Spark : Cluster Computing with Working Sets. In *USENIX conference on Hot topics in cloud computing (HotCloud)*, 2010.
188. J. Zhang and H. V. Jagadish. Lost source provenance. In *Conference on Extending Database Technology (EDBT)*, pages 311–322, 2010.
189. J. Zhang and H. V. Jagadish. Revision provenance in text documents of asynchronous collaboration. In *IEEE International Conference on Data Engineering (ICDE)*, pages 889–900, 2013.
190. W. Zhou, Q. Fei, A. Narayan, A. Haeberlen, B. T. Loo, and M. Sherr. Secure network provenance. In *ACM Symposium on Operating Systems Principles (SOPS)*, pages 295–310, 2011.
191. W. Zhou, S. Mapara, Y. Ren, Y. Li, A. Haeberlen, Z. Ives, B. T. Loo, and M. Sherr. Distributed time-aware provenance. *Proceedings of the VLDB Endowment (PVLDB)*, 6(2):49–60, 2012.
192. W. Zhou, M. Sherr, T. Tao, X. Li, B. T. Loo, and Y. Mao. Efficient querying and maintenance of network provenance at internet-scale. In *ACM Conference on the Management of Data (SIGMOD)*, pages 615–626, 2010.