Master of Science Thesis

Metadata-based Provenance

Agustín Mullin

November 2015

Advisor: Regina Motz

Instituto de Computación - Facultad de Ingeniería.
Universidad de la República. Pedeciba Informática.
Montevideo, Uruguay.

Resumen

Proveniencia de datos es el problema de explicar cómo un dato fue creado o usado. Al hacerlo se plantean muchos desafíos, como ser entender y acordar qué son los datos y los procesos que los crean y asignarles identificadores (problemas agravados en un ambiente distribuido), capturar la información de proveniencia, y sobre todo, hacer todo esto de forma transparente. Presentamos un análisis de estos desafíos y a continuación desarrollamos la mayor contribución de la tesis, un framework para la captura y registro de la proveniencia basado en metadatos para un ambiente distribuido y heterogéneo. Nuestra solución está basada en un modelo conceptual de datos, que facilita la integración de información de proveniencia originada en diferentes sistemas. Por lo tanto es posible construir un grafo de proveniencia que vincule datos producidos en diferentes sistemas. El framework define roles y responsabilidades para obtener la proveniencia de los datos. Define también un sistema para la identificación de los datos y las transformaciones que los producen así como para resolver la ubicación de los datos. También se presenta una propuesta de implementación para el repositorio de la proveniencia.

Nuestro enfoque consiste en relevar la literatura para analizar soluciones ya existentes, para primero construir una definición común de proveniencia y un modelo conceptual de datos para proveniencia basados en los trabajos relevados, y luego desarrollar nuestro framework basado en este modelo conceptual. Hemos hecho un análisis de las características de los sistemas que ofrecen proveniencia y del tipo de proveniencia que ofrecen, con el objetivo de obtener principios generales, lo que a lo mejor de nuestro conocimiento no es analizado por ningún trabajo. También hacemos explícitas algunas de las hipótesis de trabajo que son comúnmente implícitas. Además, presentamos un análisis del problema de ofrecer proveniencia de datos en un entorno distribuido con tecnologías diversas. En particular, cuando se utiliza Hadoop (el estándar de la industria para el almacenamiento y procesamiento de grandes volúmenes de datos) y las diferentes herramientas de su ecosistema, surgen nuevos retos, que analizamos y tomamos en cuenta en el diseño de nuestro framework.

Abstract

Data provenance is the pervasive problem of explaining how a datum was created or used. Doing so has many challenges, such as understanding and agreeing what data and processes are and assigning them identifiers (aggravated in a distributed environment), capturing provenance information, and all this done in a transparent manner. We present an analysis of these and in consequence develop our major contribution, a framework for provenance capture and recording based on metadata in a distributed and heterogeneous environment. Our solution is based on a conceptual data model, which facilitates the integration of provenance information originated in different systems. Thus, it is possible to construct a data provenance graph which relates data produced in different systems. The framework defines roles and responsibilities to achieve data provenance, as well as an identification scheme for data and transformations and for resolving the location for data items. We also present an implementation of the provenance repository.

Our approach consists in reviewing the literature for existing solutions, to construct first a common definition of provenance and a common conceptual data model for provenance based on the reviewed works, and then develop our framework based on this conceptual model. We have also performed an analysis of the characteristics of the systems offering provenance and the type of provenance they offer, in order to obtain more general principles, what to the best of our knowledge is not analysed by any work. We also make explicit some of the working hypothesis which are commonly implicit.

What is more, we offer an analysis of the problem of offering data provenance in a distributed environment with diverse technologies. In particular, when using Hadoop (the industry standard for big data management) and the different tools of its ecosystem, new challenges arise, which we analyse and take into account in the design of our framework.

Acknowledgements

Primero que nada quiero agradecerle a mi amada esposa Marcela por su amor y apoyo constante que me permitió llevar adelante este proyecto.

Le quiero agradecer especialmente a mi tutora Regina por motivarme, guiarme y por todo lo que me enseñó, que me ayudó a ser el profesional que soy hoy.

Por último quiero agradecerle a mis padres, mi abuela, mi hermano y mis amigos por acompañarme siempre.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Provenance is the pervasive problem of explaining how something was created or used. For example, in the context of science, data can be obtained from different measurements and processed with models to produce new data, which can be fed to other models (maybe, by a different scientific group) to produce other data. In this domain, a common requirement is to ask: who produced the data being used, as well as when and how. The how is related to which data transformations or processing steps were applied to produce it. These questions help the users enhance their interpretation of the data and assess its quality and reliability.

A classical example of data provenance in information systems is in data warehousing, where it is common to integrate data from different sources via ETL (extract, transformation and load) and data cleansing processes and to create tables or views with aggregated data. When analysing the aggregated data, common requirements are: to access the detailed information on which the aggregated data is based on, or even more, to identify the sources of a possibly faulty datum and to access the original data prior to the ETL or cleansing processes that led to the construction of that datum.

There are other examples of provenance which are not restricted to data management. For example, in an industrial process, where raw materials are used to produce some resulting products, and it is necessary to know which raw material were used to produce a given item, or vice versa, which resulting products were obtained from a raw item. Also, analysing academic publications, it is of interest to know which previous publications were referenced by a certain publication, and vice versa, which publications referenced a given one.

Offering this provenance information is challenging for many reasons: first, when data travels through different systems and is processed by different systems, it may be difficult to understand the unit of data and the unit of data transformation used by each system, not to mention, to integrate them in a common model. We have seen in the reviewed literature that different definitions of provenance exist, which include varied elements and not always refer to them with the same terms. Also, conceptual data models are seldom used, which makes it more difficult to integrate provenance information.

Also, there is a need for a data identification scheme for data items and data transformations. This is specially challenging in a diverse and distributed scenario. What is more, these identifiers should permit the user to retrieve

information about the data transformations, their executions, about the data items or the data itself. Some authors [53] use hash digests as identifiers for data, but this only works to identify data files (a.k.a. coarse granularity), not its individual inner records. To identify the individual inner records (a.k.a. fine granularity) some authors [61, 31] use their locations as identifiers, but this results in brittle solutions that break if the file is moved or reordered. Also, other authors [44] create identifiers for each record, but burden the provenance repository with saving the data and the identifiers.

What is more, provenance capture and data identification is expected to be transparent to the data transformations. We have seen that better types of provenance (finer granularity and more informative notions of provenance) can only been achieved by restricting the type of data transformations supported, thus resulting in less transparency. In order to offer the most informative type of provenance, the authors [55, 33] restrict data transformations to be declaratively defined (e.g. via a query), while other authors [46, 44] support any type of data transformations but offer a much more limited type of provenance.

In this context, we set as our objective to obtain a general, high-level, metadata-based solution, which supports the best possible notion of provenance. Thus, we have developed a framework for provenance capture and recording based on metadata. The solution is based on a conceptual data model, which facilitates the integration of concepts in the different systems. Thus, it is possible to construct a data provenance graph which relates data produced by different systems. The framework defines roles and responsibilities to achieve data provenance (in particular one for capturing and one for recording data provenance) as well as an identification scheme for data and transformations and for resolving the location for data items.

The major contribution of this thesis is the aforementioned framework together with a proposal for the implementation of the provenance repository. Other contributions are a survey of existing solutions to data provenance in different contexts, the unification of different terms used in the literature and the proposal of a definition of data provenance which includes them. Also, we have developed PCM (Provenance Conceptual Model), a conceptual data model for provenance based on the data models of the surveyed works.

Another contribution is the analysis of the characteristics of the systems offering provenance and the type of provenance offered, what to the best of our knowledge is not analysed by any work. We also make explicit some of the working hypothesis which are commonly implicit. What is more, we offer an analysis of the problem of offering data provenance in a distributed environment with diverse technologies, in particular inside the Hadoop environment (the framework for distributed storage and distributed processing of very large data sets), analysing the challenges that can appear when the different tools of the Hadoop ecosystem interact with each other.

In chapter 2, based on a review of the literature, we present a definition of data provenance, and present the characteristics of provenance as well.

In chapter 3, we present our case study and based on its requirements, we review the surveyed solutions, focusing on those which work with Hadoop. We analyse them and show they cannot be used to solve our case study.

In chapter 4 we present the analysis of the case study and propose the provenance framework as a solution. We start by explaining the need for a conceptual data model for provenance, we present the models of the reviewed

works and construct PCM (Provenance Conceptual Model). Then we analyse the difficulties for capturing and recording provenance in this scenario. We close the chapter presenting the provenance framework.

In chapter 5 we address the implementation of the provenance framework, first we review OPM and PROV-DM, two provenance models for interoperability and assess them using PCM; then we present a concrete implementation of the Provenance Repository; we present the aspects of the implementation of provenance and present how these issues are addressed in the reviewed works in a comparative summary. We show how two existing solutions for Hadoop and relational databases could be adapted to work within the framework playing the provenance capture role.

In chapter 6 we present other scenarios and show how the framework and the metadata repository proposed in chapter 5 can be adapted to satisfy the different requirements in the different use scenarios. Finally, in chapter 7 we present our conclusions and future work.

Chapter 2

Provenance Definition

When searching the reviewed literature for a definition of data provenance we find that in each case the authors are immersed in a specific domain and problem and thus present a definition of provenance that suits their requirements. This results in limited definitions. In order to study the general problem of data provenance, we need a more general definition, which we construct from the reviewed definitions.

In this chapter first we present a definition of provenance and a summary of its characteristics, we survey the different definitions for provenance given by different authors and conclude with a comprehensive definition of provenance that includes the reviewed definitions. Finally, we present a summary of the surveyed definitions comparing them to the proposed definition.

2.1 Introduction

Several definitions do exist for the provenance of a datum, also known as attribution, filiation, genealogy, lineage, parentage and pedigree. These definitions do not always consider the same elements nor refer to them using the same terms. Buneman et al. [37] -which according to Google Scholar is one of the most cited articles dealing with provenance-, define data provenance as "the description of the origins of a piece of data and the process by which it arrived in a database".

Most authors work in a particular context, where data are stored in different structures and data items are even called with different names, e.g. data can be stored in relations in the case of databases, in which case data items are generally referred to as tuples; data can be stored in files in the case of the HDFS, in which case data items are generally referred to as records; etc. We will try to refer to these in a more general way using the terms data collection and data item respectively.

We now present some general terms, then introduce each reviewed work with its definition of provenance and close the chapter giving a general definition which includes those of the the reviewed works.

In Appendix A we present a glossary with the different terms used in the reviewed literature and the definitions given by the authors.

2.1.1 Fine- and Coarse-grained provenance

Depending on how granular the provenance information is, it can be referred to as coarse-grained or fine-grained provenance. If the solution only permits to identify the source data collections from which the data item was derived, it is called coarse-grained provenance. On the other hand, if it permits to identify exactly which source data items were used to produce a given data item in the output, it is called fine-grained provenance [45, 76].

In the context of fine-grained provenance, the relation between an output data item and the input data items used to produce it is called causal relation [31].

2.1.2 Notions of provenance (Why, how and where)

In the context of fine-grained provenance it is possible for the causal relation between data items to bear a meaning, which depends on the notion of provenance being used. Cheney et al. [41] say that among the different notions of provenance proposed, the most common ones are why-, where- and how-provenance.

Buneman et al. [37] first distinguish between "why-provenance" y "where-provenance". While why-provenance explains why a data item is in the collection, which is, for a given data transformation, what pieces of input data validate the existence of an output data item; where-provenance shows where a given piece of data comes from, which "requires us to identify locations in the source data".

Why- and where-provenance are related, when we identify the where-provenance of a data item in the output, its value must have been copied from some data item from the input, which can be found in the why-provenance of the output data item.

Green et al. [55] show that why-provenance has certain limitations because even though it shows the set of contributing data items for an output data item, it does not explain how these contributed to it. It can be the case, for example, for two different output data items to have the same why-provenance and yet for them to have been created using those data items in a different way. Thus, they propose the notion of "how-provenance" which uses polynomials to represent the how the data transformation used the the input data items to produce the output data item. These polynomials are used to tag the output data items to document how they were produced.

As stated by Cheney et al. [41], how-provenance is more general than why-provenance since one can derive the why-provenance of an output tuple from its how-provenance polynomial but not the other way around.

2.1.3 Backward and Forward Tracing

The causal relation between input and output data items can be seen or queried in two ways, known as backward tracing and forward tracing.

When performing backward tracing, the provenance of a data item consists of the data transformation and the input data items used to produce it. If backward tracing is applied iteratively over those input data items, then the backward-tracing provenance of the data item in question consists of all its

origins or ancestral data products and entire processing or derivation history [63, 37, 46, 76, 77, 80].

Regarding forward tracing, the data provenance of a data item, consists of the transformations that were applied to it as well as the output data items produced by them. If applied iteratively over those output data items, then the forward-tracing provenance of a data item are all the data items that have evolved from it, also known as descendant data items [36, 61].

Backward and forward tracing of a datum are definitely related, since they are two alternative ways of seeing the same information.

2.2 Reviewed works

In this section we survey the different definitions for provenance. The reviewed works are ordered by publication year and alphabetically by first author. The process and criteria to select the reviewed works is explained in Appendix B.

Buneman et al. (2001)

Buneman et al. [37] work in the context of transformations defined as database queries (both relational and XML databases) and propose a syntactic approach to compute provenance based on analysing the query (i.e. the schema mapping). They state provenance is "essential to anyone interested in the accuracy and timeliness of the data".

The authors define provenance, lineage or pedigree, as "the description of the origins of a piece of data and the process by which it arrived in a database". Yet, they address only the part of the problem referring to the origins of a piece of data. Thus, their solution offers fine-grained provenance, distinguishing between why-provenance and where-provenance and offering a solution which works to compute the first type of provenance for general queries and computes the latter for a restricted type of queries. This limitation is due to the fact that the syntactic analysis they perform over queries cannot always determine the origin of the output result of a query (it is possible for a query to be rewritten in a different but equivalent way and for the where-provenance of a result to change, this is why the authors say where-provenance is not invariant under query rewriting).

Frew et al. (2001)

Frew et al. [53] have developed a metadata recording framework called Earth System Science Workbench (ESSW). They work in the context of science, where available data sets are used to generate higher level data products which are in turn published. In this context, the authors claim it is difficult to keep track of which are the original data sets, which ones are generated, and when, how (e.g. version of program) and by whom.

They define an "object's data lineage" [sic] as its origin and processing history. Their system permits to record coarse-grained provenance for files or data sets.

Marathe (2001)

Marathe [65] works in a context of science where data is represented in multidimensional arrays and data transformations are expressed using the declarative language AML (Array Manipulation Language), an algebra consisting of three operators that map arrays to arrays. In this context, the author presents a "lineage tracing algorithm" called SUB-pushdown for data transformations over array data which offers fine-grained provenance, in particular why-provenance.

The author defines lineage tracing as "a type of data-flow analysis that relates parts of a result array to those parts of the argument (base) arrays that have bearings on the result array parts." In particular, considering an array manipulation that computes a result array X from one or more base arrays, and considering some selected elements X' of the array X, lineage or lineage trace of X' consists of those elements of the base arrays that are required to compute X'.

Foster et al. (2002)

Foster et al. [51] work in the context of science where they say the analysis of data is a significant community activity and that as a result of this activity, communities construct, in a collaborative fashion, collections of derived data. In this context, they point out the importance of recording and discovering the relationships between data objects corresponding to the computational procedures used to derive one from another.

They have developed a prototype for a system called Chimera, which combines a virtual data catalogue, for representing data derivation procedures and derived data, with a virtual data language interpreter that translates user requests into data definition and query operations on the database.

This data catalogue permits to compute data provenance on a coarse level based on metadata, and also to perform "virtual data management" operations, for example to "re-materialize" data products that were deleted, generate data products that were defined but never created, regenerate data when data dependencies or transformation programs change, and create replicas of data products at remote locations when re-creation is more efficient than data transfer.

Provenance is not defined by the authors in this work. However, they refer to the works of Buneman and Cui which define provenance as the causal relationship between data items. Also, they point out the importance of identifying the data transformations used to create the data collections (coarse-grained granularity).

Cui et al. (2000, 2003)

Cui et al. [45, 47, 46] working in the context of data warehousing, formally define provenance (they use the term lineage) and present an algorithm to trace exact fine-grained provenance for both relational views and for non-declarative transformations. The provenance of a (view) data item is defined as "the exact set of base data items that produced" it, which coincides with the why-provenance definition as is pointed out by Buneman et al. [37].

They consider the problem of a data warehouse composed of a sequence of materialized views, where the transformations that take data through the views

"may vary from simple algebraic operations or aggregations to complex procedural code". The algorithms proposed work for non-declarative transformations, but take advantage of the schema mapping information when present. Here the separation of specification and implementation of the mappings is evidentiated.

Pancerella et al. (2003)

Pancerella et al. [72] work in the context of chemical sciences, where multi-scale research is done by passing of information from one level to the next. Also, experiments may produce new results so provenance may help to understand the accuracy and currency of scientific data. In this context, they state that one of the major bottlenecks is passing information from one level to the next in a consistent and validated manner. They have developed a system called Collaboratory for the Multi-scale Chemical Sciences (CMCS) which offers a suite of tools for managing data and metadata and visualizing "pedigree relationships" between data entries with coarse granularity.

The authors define data provenance or data pedigree as "where a piece of data came from and the process by which it arrived in the data repository" and say it is important to the accuracy and currency of data.

Bhagwat et al. (2005)

Bhagwat et al. [35] work in the scientific domain, where they say that data is collected from different sources and it is often cleansed and reformatted before it is compiled into a new database, where it is common for such newly created databases to contain new analysis or results that are derived by scientists. In this context, they say that there is information about data that is not kept in the database but that it is critical for further analysis and scientific discovery to propagate it along as data is being moved around.

With this objective, they have developed an annotation management system for relational databases in which every piece of data in a relation is assumed to have zero or more annotations associated with it and annotations are propagated along, from the source to the output, as data is being moved through by a query. The system uses the notion of where-provenance to define which annotations to propagate when a query is executed. Even though the actual origin of a new data item (i.e. its where-provenance) is identified and used by the system, this information is intentionally not recorded, but instead its where-provenance annotations are copied and assigned to the new data item.

In the system, every column of every tuple in every relation can be annotated with zero or more annotations. The term annotation is used to refer to information about data such as provenance, comments, or other types of metadata. It should be considered that it is understood by the authors that provenance is neither the causal relation (why-p) nor the location from where a data item was obtained (where-p) but instead provenance is a piece of metadata associated to each data item. As it, it needs to be propagated together with the data item. Also, note that the solution works in a context where no transformations are actually applied on the data, it is only "moved around", and therefore the system only works for SPJ queries.

The authors define the provenance of a piece of data as where that piece of data is copied or created from, and it is seen in their system as one of the many

annotations of the data. Other examples of annotations given are: the perceived accuracy or reliability of experimental results by domain experts, information about who has seen or edited a piece of data, error reports or remarks about a piece of data, and the quality or security level of a piece of data.

Green et al. (2007)

Green et al. [55] work in the context of relational databases, where data transformations are restricted to relational algebra. They show the limitation of why-provenance in that two different output data items may have the same why-provenance when they may have used those input data items in a different way. To overcome this they develop the notion of how-provenance, where a polynomial is used to show how the input data items were used to construct an output data item.

As shown by Cheney et al. [41], how-provenance is more general than why-provenance, so the latter can be computed from the former. Hence, this solution permits to compute fine-grained why-provenance.

Amsterdamer et al. (2011)

Amsterdamer et al. [33] work in the context of workflows whose transformations (referred to as modules in this work) are Pig Latin queries. In this context they propose a solution (and have built a prototype) that permits to compute fine-grained data provenance (both for backward- and forward-tracing) using metadata. Regarding the notion of provenance, their work is based on the idea of semirings developed by Green et al. [55] for the context of database, so the system permits to compute how-provenance (and therefore, why-provenance also).

They define the data transformations, which they call modules, as a Pig Latin [71] query that reads and writes in a relational schema. A distinct characteristic of these data transformations is that are allowed to have an internal state (which may be modified by inputs in previous executions of the workflow) which the provenance system takes into account.

Crawl et al. (2011)

Crawl et al. [78, 32, 44] work in the context of scientific workflows, where they have developed a workflow management systems called Kepler, which they have integrated with Hadoop, creating Kepler+Hadoop. The Kepler system permits to model and execute workflows (both through a GUI or in batch mode), the Kepler+Hadoop version presents extensions to permit the data transformations (referred to as actors in Kepler) to be MapReduce jobs. They provide a metadata based framework to compute coarse-grained provenance (both for backward and forward tracing).

Their provenance data model focuses on the workflow, its definition, its evolution and its executions. They define a workflow definition as a specification of what exists in the workflow, its entities, their parameters and the connections between the actors. Workflow evolution is a description of how the workflow definition has changed over time. Their model permits to record workflow and actor executions (a.k.a. runs), the context and data products consumed and

produced by the mentioned runs. Context is the who, what, where, when, and why that is associated with the run.

Huq et al. (2011)

Huq et al. [60] work in the context of stream data processing for data coming from sensors. This is the case where individual data items may arrive continuously in "multiple, rapid, time-varying, possibly unpredictable and unbounded streams"[34]. In this case, the transformation process is continuously executed on a subset of the data stream known as a window. Their solution permits to compute fine-grained data provenance, in particular why-provenance, since the solution "associates the chosen output tuple with the set of contributing input tuples".

Ikeda et al. (2011)

Ikeda et al. [61, 73] have built a prototype system called RAMP as an extension to Hadoop, to support the automatic capture and recording of fine-grained metadata-based provenance (both for backward and forward tracing). Their solution is for "data-oriented workflows" (directed acyclic graph where nodes denote data set transformations, and edges denote the flow of data input to and output from the transformations -each edge is annotated with a data set-), in particular for MapReduce workflows, where all transformations are pure map and reduce functions. The MapReduce functions could be implemented directly or be the result of the compilation of Pig Latin [71] (a high-level data-flow language). The case study presented is for the analysis of tweets and diggs to "gauge public opinion on movies" in batch processing of data sets (not continuous stream processing).

They define provenance the same way Cui et al. [45] do it, first for an output tuple of a transformation, as those elements of the input that contributed to its derivation, i.e. why-provenance; and then, the provenance of an output set is defined as the union of the provenance of its elements. A specific definition of provenance is defined for each transformation type (map, reduce, union and split). In this way, provenance for the workflow is defined recursively.

Akoush et al. (2013)

Akoush et al. [31] present a modified version of Hadoop -called HadoopProv- which offers capturing and recording of fine-grained why-provenance in MapReduce jobs. The authors define provenance as "the causal relationship between input and output records", which is equivalent to the definition used by Ikeda et al. [61].

2.3 General Definition and Comparative Summary

Taking these reviews into account, we give the following definition. Data provenance is the explanation of everything that happened related to the creation of data. We point out that it refers to something that has already occurred as stated by Bose et al. and Moreau et al. [36, 67]. This explanation related to the creation of data includes -but is not limited to- the following information:

I if referring to raw data, how it were created (e.g. regarding scientific measurements: the recording instrument, the instrument's operating parameters, perceived accuracy or reliability of experimental results by domain experts, etc.) [80];

II when data were processed from other data, information regarding the data transformation execution (in which context it was executed, by whom, when, using which parameters, etc.) [80, 53, 51, 72, 32];

III the specification (definition, purpose, version, developer, etc.) of the data transformation that produced the data [80, 53, 51, 72, 32, 33];

IV if the data transformation was invoked as part of another transformation or workflow, information regarding the latter [44];

V information regarding the input data, with coarse granularity and fine granularity, and explaining how these input data items were used [63, 41, 37, 45, 55].

VI other information describing the data item, such as: assumptions and criteria applied at any stage of the data product life cycle; information about who has seen or edited a piece of data; error reports or remarks about a piece of data or its quality, etc. [36, 35].

Being in possession of this information, one can approach it from different perspectives to pose different queries. The most common queries refer to backward and forward provenance tracing, which can be queried in only one step or iterating over the provenance information. That is, when performing backward provenance tracing for example, in one step one would see the information for the data items used as input by the transformation that produced it; then one could iterate over those data items, performing backward provenance tracing for them.

Other possible queries over this information are: to identify all data items produced with a certain version of a data transformation; to identify data produced by transformations run by a certain user or under certain conditions; and integrating it with forward tracing, to identify all descendants of these data items.

In table 2.1 we show which aspects of the definition are considered by each of the reviewed works, showing also the granularity of the solution, the notion of provenance and if the solution offers backward or forward tracing. In this analysis we can observe the differences in the reviewed definitions. Also, we can see the one element that is present in all definitions of provenance is the causal relation between input and output.

Reviewed Works	I&VI	II	III	IV	V	Gran.	Notion of Prov.	Bw / Fw Tracing
Buneman et al. [37]		x	x		x	Fine	Why-p, Where-p	Bw
Frew et al. [53]		x	x		x	Coarse	-	Bw
Marathe [65]					x	Fine	Why-p	Bw
Foster et al. [51]		x	x		x	Coarse	-	Bw
Cui et al. [45]					x	Fine	Why-p	Bw
Pancerella [72]	x	x	x		x	Coarse	-	Bw
Bhagwat et al. [35]	x				x	Fine	Where-p	Bw
Green et al. [55]			x		x	Fine	How-p, Why-p	Bw
Amsterdamer et al. [33]			x	x	x	Fine	How-p, Why-p	Both
Crawl et al. [44]	x	x	x	x	x	Coarse	-	Both
Huq et al. [60]					x	Fine	Why-p	Bw
Ikeda et al. [61]					x	Fine	Why-p	Both
Akoush et al. [31]					x	Fine	Why-p	Both

Table 2.1: Comparative summary of reviewed definitions.

Chapter 3

Case Study and Available Solutions

In this chapter we present our case study -of provenance requirements in a Hadoop (an open-source framework for "reliable, scalable, distributed computing" [14]) environment- and based on these requirements we review the surveyed solutions for Hadoop to analyse if they could be used to address them. We show that none of them address all of aspects the scenario.

3.1 Case Study

We present a case study with the objective of illustrating the different situations and challenges that can arise when tracing provenance in a Hadoop environment. Hadoop is an open-source framework for "reliable, scalable, distributed computing". The core of Hadoop consists of Hadoop Distributed File System, a.k.a. HDFS, and Hadoop MapReduce (parallel processing of large data sets) [14]. Yet, when referring to Hadoop we will consider also Hadoop's Ecosystem, which includes several widely used frameworks and tools built over Hadoop, which offer different abstractions and paradigms for storing and processing data.

The problem we consider is that of updating quality attributes of derived data when the quality attributes of their ancestral data are updated.

The case study consists of a distributed environment, composed of a Hadoop system used to summarize large volumes of data, and then the summarized data are copied to a relational database where, after being integrated with other data, are used for decision making.

In figure 3.1 we present the case study. It is composed of a Hadoop system which has four data files, A and B where data are loaded from different sources, and C where data are integrated from A and B. The data in C is then summarized in D. The summarized data in D are copied to the relational database, in particular to the relation D'. These data transformations can be performed using any of the frameworks in the Hadoop ecosystem.

Once data arrives to D', using the data available in the relation E, the data in the relation F are created, which is then input to a decision making process.

The original data loaded in Hadoop (files A and B) are loaded together with quality attributes. Based on these, the quality attributes of C and of the

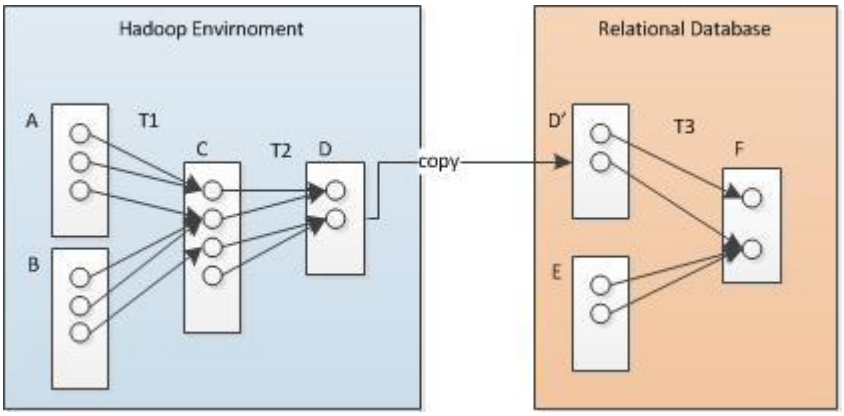

Figure 3.1: Different data transformations running in different environments. The arrows represent the causal relation between input and output data items of each transformation execution.

summary data (D) are calculated. And then the quality of data in the relation F is calculated. Then decisions are based on the calculated quality of the latter.

So when a notification rectifying the quality attributes of the original data in Hadoop (either A or B, with fine granularity), it is of vital importance to identify the data items in F which were derived (indirectly) from them and to update their quality attributes so better decisions can be made. This is expected to be performed as fast as possible.

For each of the processing steps it is also necessary to know which data transformation was performed. Another requirement is that the system should be able to communicate the provenance information it computes.

It is desirable that provenance capture and recording is as much transparent as possible to the developers of the transformations and workflows.

3.2 Analysis of existing solutions

In this section we study the reviewed solutions which are designed for the Hadoop environment, select two of them as a reference and analyse how well they satisfy the requirements.

We present in table 3.1 a summary of the surveyed solutions which work in the Hadoop data management context. First, as for the definition of provenance, most of the selected solutions restrict the concept of provenance to the causal relation between input and output records, so no information regarding transformation executions nor context is captured by them. The exception is the solution presented by Crawl et al. [44] which puts great emphasis on it, especially on the concept of workflow. As for what they offer, most of the surveyed solutions offer fine-grained provenance, except for that Crawl et al. which offers coarse-grained provenance. Yet, it also offers information about the workflow that produced the data and context information of its execution.

Most of the selected solutions are based on metadata and one follows a hybrid approach (none of the solutions is based on a tracing procedure). Also, regarding what is included in the scope of the solution, most solutions address

	Amsterdamer et al.	Ikeda et al.	Akoush et al.	Crawl et al.
Provenance definition	Causal relationship	Causal relationship	Causal relationship	Causal relationship and workflow executions
What they offer	Fine-grained, how-p and why-p	Fine-grained, why-p	Fine-grained, why-p	Coarse-grained & WF Executions and Specification
Metadata or Tracing procedure	Metadata	Metadata	Hybrid	Metadata
Scope of the Solution	Not specified	Only provenance capture. Locations used as identifiers.	Only provenance capture. Locations used as identifiers.	Provenance capture and Data and Process identification.
Hadoop ecosystem	Pig and MapReduce jobs.	Pure MapReduce jobs.	Pure MapReduce jobs.	MapReduce jobs executed from Kepler.
Only inside Hadoop?	Only inside Hadoop.	Only inside Hadoop.	Only inside Hadoop.	Distributed environment. Centralized metadata repository.

Table 3.1: Summary of the surveyed solutions which work in Hadoop.

only the problem of provenance capture, and only that of Crawl et al. address the problem of identification of data items and data transformation executions.

Finally, we show the tools and type of data transformations inside Hadoop that are supported by the solution, for instance, there are solutions for data transformations defined using Pig Latin, for pure MapReduce jobs and for general MapReduce jobs; and if the solution permits to integrate provenance metadata from other systems.

We will focus on the solutions designed by Crawl et al. [44] and that by Ikeda et al. [61] since these are the ones with a more detailed explanation of their implementation.

We will not focus on the solution developed by Amsterdamer et al. [33], since the authors omit many aspects of their solution, such as how are the data items identified and how is metadata stored in the file system. The solution developed by Akoush et al. [31] also offers little details of its implementation, but seems to be an improved version of that developed by Ikeda et al. [61].

Solutions of Ikeda et al. and Crawl et al.

Based on the previous analysis done in this chapter, we now go over the two selected solutions. As for the definition of provenance, Ikeda et al. [61] restrict the concept of provenance to the association between input and output records, so no information regarding neither transformation executions nor context is captured.

Alternatively, Crawl et al. [44] include the workflow definition and run in their provenance requirements. Another difference is that Ikeda et al. offer fine-grained provenance while Crawl et al. offer coarse-grained provenance.

Regarding the scope of their solutions, Ikeda et al. only address the capture and recording of provenance based on a given identification scheme or on a default one (the data item location). The authors do not address the issue of identifying data items. Also, the proposed solution does not address the maintenance of the metadata, thus the solution is heavily dependent on the form data is stored in the file system, even the way it is ordered inside the files, e.g. Hive tables (i.e. HDFS files) partitions can wreak the system.

On the other hand, Crawl et al. explicitly address the issue of data and process identifiers creation. In their solution, data are always moved around with their identifiers, which is transparently handled by their framework.

Regarding the frameworks supported in Hadoop and the integration with other systems, Ikeda et al. address provenance for MapReduce jobs only, which would work for Pig Latin programs and HiveQL queries which compile to MapReduce jobs. However, the metadata could break should files be sorted or partitioned, which are common Hive operations. So other frameworks are not really supported. In this case, the solution developed by Crawl et al. is even more restrictive because it only supports MapReduce jobs run from inside the Kepler system.

Lastly, as for the integration of provenance with other systems, Ikeda et al. specifically address the possibility of tracing provenance through a sequence of MapReduce jobs, but do not mention the possibility of tracing provenance through different systems. This would be difficult since no identification scheme is defined. In contrast, Crawl et al. consider a scenario where data can be moved inside Hadoop to be processed and then moved out to continue its processing, while provenance is traced through all the workflow. They follow a metadata approach to provenance which is recorded in a centralized manner. This solution does not support systems which are not able to record provenance information, but this is not really a problem because they restrict themselves to data transformations executed from Kepler.

The two reviewed works address only a small part of the potential general problem. Ikeda et al. cover more situations since their solution captures provenance for any MapReduce job running in Hadoop, while that of Crawl et al. works only when the jobs are run from a Kepler workflow. Another big difference is that Crawl et al. address the problem of assigning identifiers to data items, which then permits tracing provenance through different systems, a possibility not offered by Ikeda et al. Since data items are assigned identifiers, the two must be handled together, which implies an extra difficulty to provenance capture, which results in Crawl et al. offering only coarse-grained provenance, while Ikeda et al. offer fine-grained provenance.

3.3 Conclusions

We have presented our case study of a distributed scenario including a Hadoop environment. Of the reviewed works, we have selected those which offer provenance in a Hadoop environment and analysed how well they satisfy the requirements of our case study. We cannot say that one solution is completely better than other, because while one may cover in a better way some of the dimensions of the problem, it is outperformed in another dimension by the other solution. Yet, none of them addresses all of aspects the scenario, so in the following chapter we present a framework for provenance capture and recording which satisfies the requirements of the case study.

Chapter 4

A General Framework for Provenance

In this chapter we present an analysis and a solution for the case study. In order to address the requirements we start by developing a conceptual data model of provenance, representing the concepts and relations to which we refer. Then we present an analysis of the case study and close the chapter presenting a framework for capturing and recording data provenance which satisfies the requirements of the case study.

4.1 A Provenance Conceptual Model

In order to address the requirements of provenance we need a model to represent the concepts and relations to which we refer. This model should be a conceptual model, which is a "concise description of the data requirements of the users and includes detailed descriptions of the entity types, relationships, and constraints" [49]. It should be expressed using a high-level data model and should not include implementation details.

We distinguish between conceptual, logical and physical design. In database design, once the conceptual data model is designed, the following step is the logical design, the process of transforming the conceptual schema into the implementation data model, such as the relational or object-oriented data model, or an HBase table data model. Then comes the physical design, during which the internal storage structures, indexes, access paths, and file organizations for the database files are specified [49, 64].

In this section we start by presenting the data models of the reviewed works. Here we find that the authors rarely present conceptual data models, instead they generally present only the logical (relational) model or sometimes they do not present a model at all. When there is no conceptual data model presented we construct it by making the abstraction from the logical model, so next we can build PCM (Provenance Conceptual Model), a conceptual model which includes the concepts and relations of the reviewed models.

4.1.1 Reviewed works

In this section we review the different provenance data models of the surveyed works. The reviewed works are ordered by publication year and alphabetically by first author.

Buneman et al. (2001)

Buneman et al. [37] define provenance, lineage or pedigree, as "the description of the origins of a piece of data and the process by which it arrived in a database". Yet, they address only the part of the problem referring to the origins of a piece of data. They present no conceptual model for provenance. However, their model consists of the causal relation between input and output data items, expressing both why- and where-provenance.

Frew et al. (2001)

Frew et al. [53] have developed a metadata recording framework called Earth System Science Workbench (ESSW). They present no conceptual model but instead present the system's logical relational schema (figure 4.1). Their model is difficult to comprehend because the terms are vaguely defined and are explained using the logical model.

The authors use the general concept of "science object" which is defined as "data and process", and works as an umbrella-term that includes everything in their model. Each science object is recorded in the ScienceObject table and in its specific table.

One core concept in their data model is that of "model". Although not explicitly defined by the authors, it is said that it can be modelled as a DAG (Directed Acyclic Graph) and that it can have many instances called experiments. Thus the model is a specification of a workflow of data transformations while the experiments are the executions of the workflows.

An experiment (a.k.a. a model instance) is composed of processing steps, processing activities or experiment steps, which are a data transformation execution.

Using the umbrella-term of science object, their system tracks how these science objects are linked and can record additional metadata for them. (The association recorded is that between science objects, which permits to declare for instance the relation between data as input or output to transformations. The metadata element for a science object is an XML document holding all metadata elements and values for it. No specification for these XML files is provided.

Marathe (2001)

Marathe [65] defines lineage tracing as "a type of data-flow analysis that relates parts of a result array to those parts of the argument (base) arrays that have bearings on the result array parts". Thus, no conceptual data model for provenance is defined. Their model consists of the causal relation between input and output data items, expressing why-provenance.

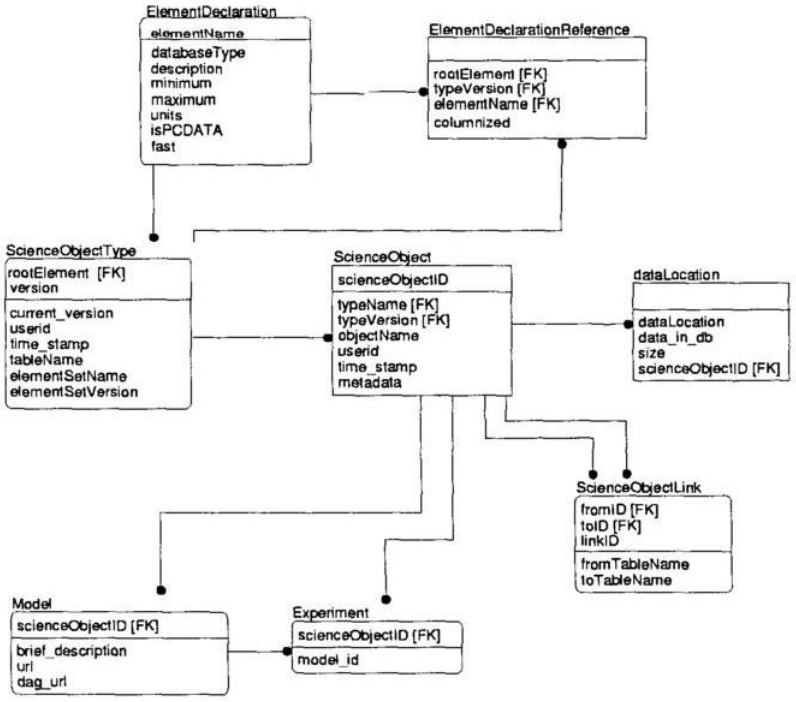

Figure 4.1: Earth System Science Workbench (ESSW) database relational schema [53].

Foster et al. (2002)

The Chimera [51] virtual data schema defines and records a set of relations which follow two objectives: 1) to formalize descriptions of how a program can be invoked, and 2) to record its actual invocations. The latter are the ones of interest regarding provenance.

The authors present a conceptual data model expressed using UML which we can see in figure 4.2. In this data model, transformations are registered (i.e. executable programs, with the information needed to invoke it), along with the derivations (i.e. the several executions of each program, together with the values for the parameters, execution time, etc.). Transformations are the equivalent to data transformation specifications and derivations are the equivalent to data transformation executions. In description of the schema.

The authors also define a data object as a named entity which was consumed or produced by a derivation (which can be files, relations or objects, granularity is coarse).

In their data model, they distinguish logical from physical transformations, where the logical transformation is the specification while the physical transformation describes where it can be accessed. Physical transformations are associated to one logical one, while a logical transformation can be associated to many physical ones. This information is important for the virtual data management operations the system offers, but not for provenance.

Derivations, the execution of the transformations, are associated to the logical transformations but not to the physical ones. Logical transformations have formal arguments defined, while the derivations have actual arguments defined, which correspond to the formal ones. An actual argument can be either a LFN (logical file name) or the value of a non-file parameter.

Cui et al. (2000, 2003)

Cui et al. [45, 47, 46] define provenance of a data item as "the exact set of base data items that produced" it. No conceptual model is explicitly presented, even though is a subjacent model in their solution.

The way they define it, provenance results contain the tables and tuples in the corresponding source that produced the result tuple for which provenance was queried. Thus, it is the causal relation between input and output data items.

Yet, the authors define a transformation as "any procedure that takes data sets as input and produces data sets as output", which may vary from "simple algebraic operations or aggregations to complex procedural code". However, the information regarding which transformation produced a given data item is not of interest in their work.

These transformations can compose a workflow or a sequence of a directed acyclic graph (DAG) of transformations. The authors define provenance, i.e. the causal relation between input and output, for one transformation and their algorithm works in that basis. Yet, when that transformation composes a workflow, the provenance of the sequence is computed by applying the same algorithm in an iterative fashion.

Since the transformation is not part of their provenance definition, no emphasis is given to it, and the specifications and executions of the data transfor-

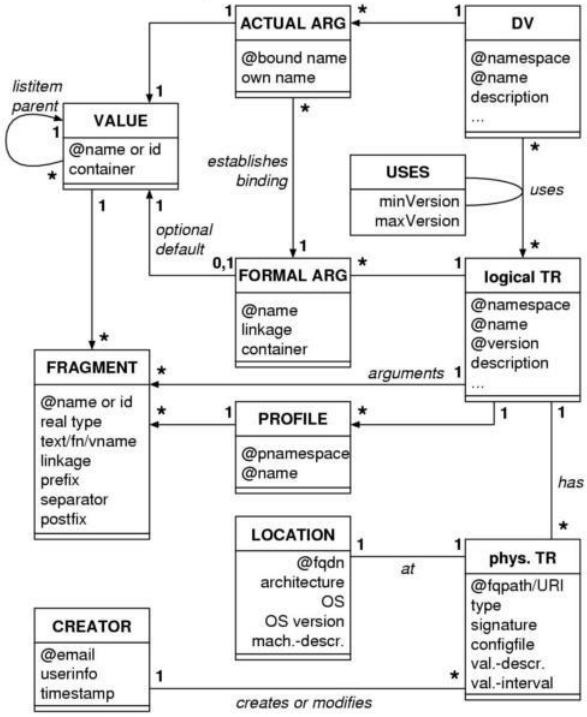

Figure 4.2: Chimera UML description of the schema [51].

mations (and workflows) are not distinguished.

Pancerella et al. (2003)

Pancerella et al. [72] define data provenance or data pedigree as "where a piece of data came from and the process by which it arrived in the data repository" and focus their data model on the description of files (data sets). Their system is basically a repository of metadata used to describe files and their relations to projects and to other data sets. The authors do not present a conceptual data model but only use attributes to tag files.

The metadata they use is based on Dublin Core, from which they employ the following elements to describe the data files: Title, Creator, Subject, Description, Publisher, Contributor, Date, Type, Format, Source, Language, Relation, and Rights. Also, they use the following Dublin Core Element Refinements: Abstract, Created, Valid, Available, Issued, Modified, Is Version Of, Has Version, Is Replaced By, Replaces, Is Referenced By, and Has References.

What is more, they have developed some extensions with the object of: enabling chemistry-specific searching, defining the relationship of scientific data to projects and related inputs and outputs (i.e. computing provenance, in particular why-provenance) and for scientific publication and peer review annotations. The metadata attributes developed for provenance are:

- Has Inputs: Used to define the input files needed to recreate the data results.
- Has Outputs: Used to define the output files created by the process.
- Is Part Of Project: Used to reference a project to a data collection or resource.
- Is Sanctioned By: Used to reference one or more review boards or organizations that have approved or "blessed" the data.

Regarding the last two annotations (Is Part Of Project and Is Sanctioned By), no definition is given of the concepts of project, review board or organizations, and it is not defined as an entity with possible elements either.

Bhagwat et al. (2005)

Bhagwat et al. [35] state that the provenance of a datum is something that can be described in one of its annotations. The authors define the provenance of a piece of data as where that piece of data is copied or created from, yet this definition is too vague to define a provenance data model and to define the structure of the mentioned provenance annotations.

The authors give the same treatment as that given to provenance to: the perceived accuracy or reliability of experimental results by domain experts, information about who has seen or edited a piece of data, error reports or remarks about a piece of data, and the quality or security level of a piece of data. No model is provided for these either.

Green et al. (2007)

Green et al. [55] define provenance for a data item as a polynomial that represents not only which data items were used to create it, but also how they were used. No conceptual data model is presented.

Amsterdamer et al. (2011)

Amsterdamer et al. [33] present a conceptual data model which permits to express fine-grain records, associate them to module invocations, to both workflows or modules input and output records, and to other records and state records via operator nodes (i.e. how-provenance). They define data transformations, which they call modules. In their model, multiple modules may be combined in a workflow, which is defined by "a Directed Acyclic Graph (DAG) in which every node is annotated with a module identifier (name), and edges pass data between modules". A workflow execution is defined as a sequence of module invocations. The logical flow of a workflow execution and its input-output relations is defined by them as coarse-grained provenance. Then they define fine-grained provenance that allows "zooming-into" modules to observe the features of the execution, such as the database's state of a module, operations performed, and computational dependencies between data.

Over this model, the system offers the operation of "zooming-out" of a module invocation provenance graph, which returns another graph where the internal nodes of a module are omitted and only its input and output nodes are returned. In this zoomed-out graph, the notion of provenance offered is simply of why-provenance, and not how-provenance, since no information is given of how the input records were used to compute the output records. It just shows which input records were used but still with fine granularity (using the literature term). However, the authors refer to this zoomed-out graph as a coarse-grained provenance graph.

The idea of "zooming-out" is similar to the overlapping accounts of the same past execution offered by the Open Provenance Model [67], which aims to offer different levels of explanation for such execution. The difference is that the OPM is more general, permitting the user to hide complete data transformations, while the zoom-out in this work permits to hide only the details occurring inside a data transformation.

Crawl et al. (2011)

Kepler's [78, 32, 44, 28] conceptual data model is based on an actor-oriented paradigm, where the workflow is composed of actors. An actor is the implementation of a specific function that needs to be performed. Each actor performs a specific independent task that can be implemented as atomic or composite (a.k.a. sub-workflows, are composed of atomic actors bundled together to perform complex operations). Communication between actors takes place via tokens that contain both data and messages.

The order of execution of actors in the workflow is specified by an independent entity called the director. Directors specify what flows as tokens between the actors; how the communication between the actors is achieved; when actors execute (a.k.a. fire); and when the overall workflow can stop execution.

Finally, a workflow is a linked set of components -the Actors- that may execute under different Models of Computations -the directors-.

Their provenance system records the values of all data passed between actors during workflow execution and the dependencies between data and actor executions.

The provenance information is recorded in a relational database together with the values of the data items. The logical relational schema consists of the following tables, which are designed using terminology of OPM, representing tokens and transfer events as OPM Artifacts and actor executions as OPM Processes:

artifact(Aid,V,C) denotes that artifact Aid has a value V and the checksum of the value is C.

compress(C,M) denotes that a checksum C has a compressed value M.

actor(E,T,R) denotes that an actor named E has a number T for workflow run R.

dependency(Aid,Pid,D) denotes that artifact Aid was read or written, specified by D, by process Pid.

As can be observed from the schema, data items (artifacts) are identified with fine granularity, but all that is said of them is that they were either input or output to data transformations (an actor) but it is not possible to declare a causal relationship between data items.

The authors use OPM terms to describe their model, which may be used as mapping to export it to OPM.

Huq et al. (2011)

Huq et al. [60] define provenance of an output tuple as the set of contributing input tuples (fine grained provenance, why-p notion). Their model also includes the coarse provenance metadata, which refers to the "processing elements" (i.e. the data transformations) and its sources and destinations.

The concept of data transformation execution does not apply to this context because the transformation process is continuously executed as data items arrive continuously in streams. As the authors point out, this coarse-grained provenance is documented during the setup of a processing element.

The term workflow is used in the article but is not defined by the authors and apparently it is not part of the provenance model. The workflow seems to be only a sequence of processing elements.

Ikeda et al. (2011)

Ikeda et al. [61, 73] define "data-oriented workflows" as directed acyclic graph where nodes denote data set transformations and edges denote the flow of data input to and output from the transformations. They focus on a special case of such workflows, which they call generalized map and reduce workflows (GMRWs) in which all transformations are either map or reduce functions.

The authors define provenance the same way Cui et al. [45] do it, first for one output tuple of a transformation, as those elements of the input that

contributed to its derivation, i.e. why-provenance; and then, the provenance of an output set is defined as the union of the provenance of its elements. Using this definition, provenance for a workflow is defined recursively.

Regarding the data transformations and workflows, they are defined and taken into account in the provenance tracing algorithm, but information about them is not considered as relevant provenance information that should be given to the final user.

Akoush et al. (2013)

Akoush et al. [31] work in a Hadoop environment where data transformations are MapReduce jobs and a workflow is a series of MapReduce jobs. They define data provenance of MapReduce jobs as the causal relationship between input and output records; and say the same definition is extended to "link provenance information across an entire workflow".

4.1.2 Defining a Conceptual Data Model

Based on the reviewed works, their definitions of provenance and the data models they define, we define PCM (Provenance Conceptual Model), a conceptual data model for provenance to include all the different concepts and relations considered by them which we considered were relevant and well defined. We present PCM in figure 4.3 using UML. We also mention the elements that were not included. This model serves not only as a summary of the reviewed literature but also as a requirement against which to assess existing provenance models.

Different authors work with coarse or fine-grained provenance, for which we respectively define the context-agnostic concepts of Data Collection and Data Item. A Data Collection is a group of Data Items that is understood as an entity by the user. Regarding Data Collections and Data Items, there is a need to declare things about them [80, 72, 35].

Also, there is a need to identify data transformations and their inputs and outputs [53]. Also, some authors distinguish between the specification and the executions (a.k.a. runs) of the data transformations [51]. The executions of the data transformations can include information such as the parameters used, its execution time, etc. Other expect to know also the context in which the transformation was executed, understood as "the who, what, where, when, and why that is associated with the run" [32].

We define the context-agnostic concept of data transformation as a general transformation that takes data items as input to produce output data items. We will distinguish two aspects of a data transformation: its specification (its name, purpose, version, its definition, etc.) and its executions or runs.

What is more, several authors consider sequences of transformations or workflows. So we define the context-agnostic concept of workflow which is a composition of data transformations. We consider it to be a directed acyclic graph (DAG) of data transformations, since this is the most common approach in the literature [45, 33, 61, 60, 31].

We point out that we consider a sequence of data transformations to be a workflow only if it has been explicitly defined that way by the user. This way, it is possible to talk about the purpose of the workflow and also of its

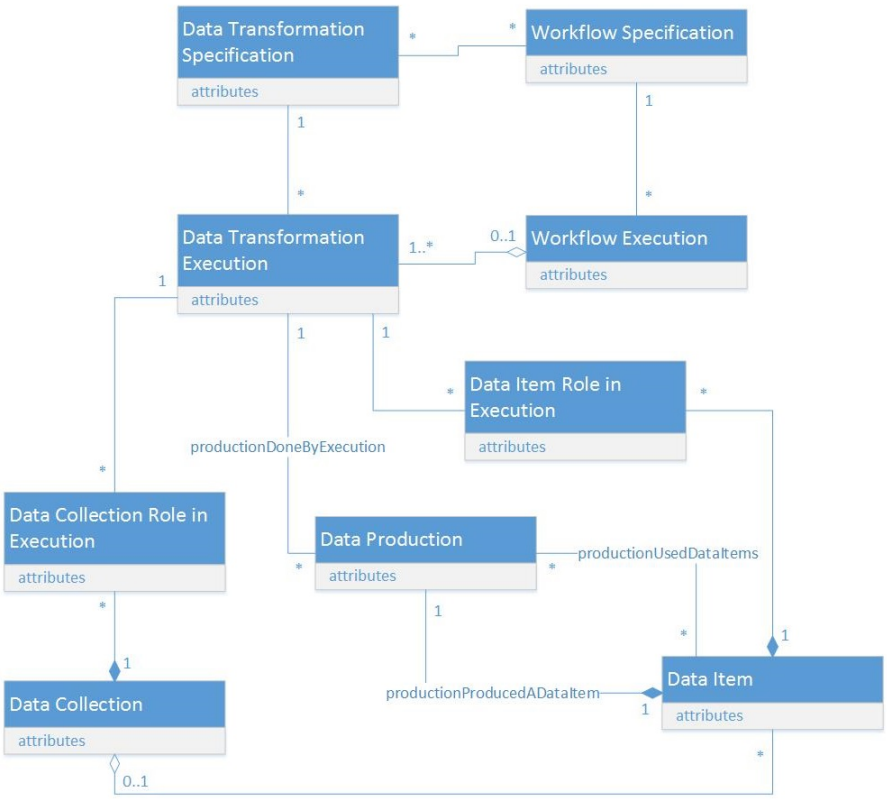

Figure 4.3: Provenance Conceptual Model.

different versions. Regarding workflows we will distinguish its specifications (which include its versions) and its executions.

Requiring that the workflow be defined by the user in order for a sequence of data transformations to be considered a workflow restricts in no way the possibilities of provenance tracing.

In the context of fine-grained provenance, it is requested to at least identify the set of contributing for a data item (known as why-provenance) [65, 45], but other authors require to incorporate a semantics to the relation between input and output data items, for example where-provenance [37] or how-provenance [55].

To represent the semantics of the creation of a data item, we include the concept of data production, which relates to the transformation execution it belongs to and to its input data items and explains how they were used to produce the output data item.

As for the elements not included in the model, they are those who refer to who executed the transformations. Since this element is only mentioned by two authors, Pancerella et al. [72] and Crawl et al. [44], and not much emphasis is put in it, we opt not to include it.

4.2 Analysis for a General Provenance Framework

In this section we present an analysis of the general aspects of the provenance problem as well as the difficulties that arise in this scenario, specifically related to the Hadoop ecosystem and to having a distributed scenario.

4.2.1 Transparent Provenance and User Intervention

We will start by stating explicitly a common objective of the reviewed works which regards to the required user intervention for the capture and recording of data provenance. The ideal is a system which can capture and record provenance in a completely transparent manner and without imposing any restrictions either to the user or the developer. However, there are limits to what can be captured automatically.

On the one hand, the causal relation between input and output of a data transformation execution can be expected to be captured automatically. Yet, fine grained provenance can only be obtained automatically when certain conditions hold. What is more, how and where provenance can only be obtained with even more restrictions. Therefore, in a distributed environment with different systems capturing provenance and pursuing this objective, it may not be possible for all of them to offer the same notion of provenance not even provenance information with the same granularity.

On the other hand, there are some aspects of provenance that cannot be expected to be captured in an automatic and transparent way, such as the name, description or objective of the data transformations. In order to have information about the transformations that produced the data opens the door to a range of additional requirements.

First, it is necessary to define what a transformation is in the context of work, for it to be possible to identify and register them, identify them when

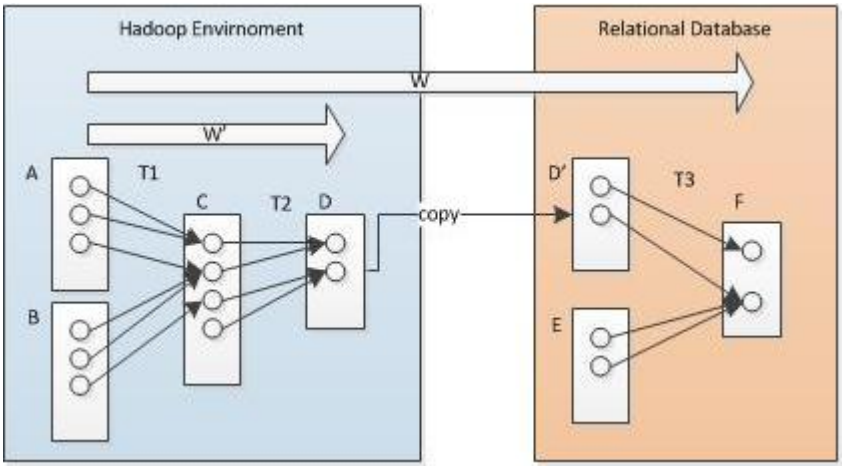

Figure 4.4: Different conceptions of a transformation or a workflow.

they execute and associate them to the data they create, e.g. in Hadoop, both a map and a reduce function could be thought of as individual transformations (e.g. Crawl et al. [78]), also a MapReduce job (e.g. Ikeda et al. [61]) or a sequence of MapReduce jobs could also be thought of as a single transformation as well. The difference between both ends of the spectrum is the thing being traced and the granularity of information the user receives. What is more, the granularity of what is considered a transformation could be fixed but it could also be variably defined by the user. This is something that needs to be defined within the functional requirements.

Besides considering the data transformations, in the reviewed literature, provenance systems usually provide support for workflows, i.e. a user-defined and organized sequence of transformations. The workflow element can be seen as a high level abstraction of the transformations and can provide high level provenance information, an idea which is covered by the Open Provenance Model [67] with the idea of multiple levels of description. So, if the provenance system should be able to capture provenance for workflows as well, it is necessary to define what is understood by a workflow.

We point out that just because two transformations take place one after another, it does not necessarily imply that the data created by the second one should be identified as created by a workflow. In figure 4.4 we analyse the case study, where for example, one criterion could be to define the whole sequence of transformations as a workflow (W), while another criterion could be to consider only the first two transformations as a workflow (W'). There is no correct or incorrect answer, the only thing is that for a data item to be identified as having been created by a workflow, we understand it should have been explicitly stated by the user that the sequence of transformations that created that data item are indeed part of a larger entity.

As a consequence, it is necessary to have a register of the transformations and workflows available in the system. This permits to associate the created data to transformations the user can recognize, and maintain additional information describing these transformations. Some of the reviewed works require

versioning of the transformations and workflows [72, 44]. Having a registry of transformations can facilitate the versioning by offering the user a place to declare that one transformation is a newer version of another and declare in what they differ, and then to distinguish data created by one or other version of the transformation.

4.2.2 Metadata or Tracing Procedure

Simmhan et al. [76] distinguish the two major approaches for obtaining provenance information between metadata approaches and tracing procedure (actually, he refers to them as "annotations" or "inversion" approaches respectively). We will use the terms metadata approach instead of annotations because it is more common in the literature and tracing procedure instead of inversion, because a transformation may or not have an inverse, but a tracing procedure can be designed independently of that, as pointed out by Cui et al. [45].

In the metadata approach, metadata comprising of the derivation history of a data product is collected as annotations and descriptions about source data and processes when the transformation takes place. This is an eager form of representation in that provenance is pre-computed and readily usable as metadata. On the other hand, the tracing procedure implies that certain algorithm must be run in order to find the input data supplied to derive the output data (the tracing procedure can sometimes imply to rerun the original transformation, e.g. Cui et al.'s solution [46]).

The solutions which are based on a tracing procedure also depend on some metadata, such as recording the mapping between two schemas, e.g., the query that relates two relations, or properties of the transformation, etc. However, these are referred to as not metadata-based in opposition to those which record metadata for every data item for which provenance is to be computed. The metadata stored by these solutions is sometimes called process-oriented [76] or coarse-grained provenance metadata [33, 60].

Simmhan et al. [76] argue that metadata should be used to provide data provenance at a coarse level, but for a finer granularity, the amount of metadata required would be far too large. Alternatively, the metadata required to apply a tracing procedure is much more compact.

Besides of the disk space used by one approach or the other, another aspect is the processing overhead. When a metadata approach is followed, transformations are burdened with collecting and recording the metadata, which is hence readily available to query later. On a tracing procedure approach, the transformation is freed from this, but instead the burden falls on the tracing procedure which must execute some algorithm every time provenance for a datum is queried.

There is therefore a trade-off between obtaining provenance quickly and running the transformations quickly. This is schematically represented in table 4.1.

Also, there are some solutions that follow an intermediate path between these two, to which we refer as a hybrid approach. The intention of these is to offer a solution of compromise between the metadata and the tracing procedure approaches.

When opting between these options, the first thing to consider is the requirements for provenance: must the system be able to answer provenance queries

Approach	Transformation execution	Provenance computation
Metadata	slow (penalized)	fast (metadata readily available)
Tracing procedure	fast (not penalized)	slow (implies running tracing procedure)

Table 4.1: Trade-off between obtaining provenance quickly and running the transformations quickly.

fast and is the capture overhead not a problem? Or the system running the data transformations is a real time system and is there no requirement about the speed of provenance answers?

Yet another aspect to consider is the possibilities of the system running the transformations, since it may not be possible to choose. For instance, it may be the case that a tracing procedure cannot be implemented because, due to the characteristics of the transformation, it would imply rerunning the transformation which would be prohibitively expensive. Alternatively, the system running the transformations is a closed one but still natively offer a tracing procedure. In that case, should one opt for a metadata approach, that tracing procedure should be invoked systematically to materialize that metadata.

To conclude, given the requirements for fast provenance answering one would opt for a metadata approach. However, in an environment where there exist different systems with diverse characteristics, it is better to leave the door open for systems limited to other approaches.

4.2.3 Identifying Data Items

Provenance is a kind a metadata, and as such it could be added directly over the data or alternatively managed separately of it. There are several arguments in favour of managing it separately, such as not modifying the schemas of the problem specific transformations, which is related to having a provenance solution which is independent from the problem specific solutions. Another argument in favour of the latter is that provenance information may need to be recorded in different ways, according to the information requirements and the ways it is going to be accessed.

In order to manage provenance metadata separately from the actual data, a form of identification for the data is needed, and therefore all of the reviewed works implement some strategy to achieve this.

When offering coarse-grained granularity, the reviewed solutions assign a unique identifier to each file and address as separate problems the identification of the file from managing its location. Such is the case of Foster et al. [51] and Pancerella et al. [72]. Frew et al. [53] use a MD5 hash digest as the identifier for the files and maintain the id-location associations.

When offering fine-grained granularity, every data item is either assumed to have an identifier or assigned one. In the context of databases, the reviewed authors [37, 65, 55] assume that tuples (i.e. data items) can be identified inside a relation (i.e. a data collection), while the relations are uniquely identified by their name inside the database. Also, in the context of MapReduce jobs, Ikeda

et al. [61, 73] and Akoush et al. [31] identify data records by the name of the file they are in together with the position they hold inside that file. Similarly, Crawl et al. [78, 32, 44] assign each record an identifier composed of the identifier of the MapReduce task that created it and a unique identifier inside that particular task. The difference between the two lies in using as part of the identifier the created file or the executing task.

Therefore, for a system to offer provenance metadata there is need for a data identifying scheme. The core of the provenance problem is capturing the required information when a data item is created, yet, in a metadata approach, the provenance information be recorded only once the created data items have been given identifiers.

An important issue to address is how to identify the data items, and if the data identification method should be provided by the provenance framework or it should be dealt outside of it. Some authors assume the existence of identifiers while others -working in a context where no natural identifiers exist- address the issue explicitly. In a distributed environment, one cannot assume the existence of a data identifying scheme. Also, if this identifying scheme existed, one could not assume it creates uniform nor globally unique identifiers.

Another issue is whether the provenance framework should be aware of data sets being moved around or altered or this is beyond its scope. What should the relation between data identifiers and data locators be? In the solution developed by Ikeda et al. [61], a provenance file is created for every output data file. Should the file be moved or renamed via file system operations, the metadata registry becomes inconsistent. What is more, the position (offset) of a record in the file is used as local identifier within that file. So, if the data files are altered, e.g. sorted, partitioned, etc., the provenance metadata based on file offsets also becomes inconsistent. We can see that in this solution unique locators are used as trivial identifiers, so it is necessary to maintain the provenance metadata when data are moved. Therefore, it seems convenient to maintain separately the identifiers of data from its location.

4.2.4 Hadoop ecosystem

Hadoop is an open-source framework for "reliable, scalable, distributed computing". The core of Hadoop consists of Hadoop Distributed File System, a.k.a. HDFS, and Hadoop MapReduce (parallel processing of large data sets) [14]. Yet, when referring to Hadoop it is common to consider also Hadoop's Ecosystem, which includes the frameworks and tools built over Hadoop. The Hadoop Ecosystem includes different tools to process and record data, which are widely used. These tools, developed by different groups or companies (Pig: Yahoo!, Hive & Presto: Facebook, HBase: developed after Google's Big Table), follow different paradigms and sometimes overlap in the functionalities they offer. What is more, these tools are able to intercommunicate and share data, which weaves an intricate web of relations and dependencies between them, which poses several challenges for provenance capture and recording.

For a system to capture provenance inside Hadoop, it is necessary to define which of these frameworks (or none, it could be just HDFS and MapReduce) are going to be supported, i.e. which of these are going to be used to record and process data while expecting provenance capture and recording, since specific solutions (or modules) need to be developed to address each one of them.

It must be taken into account that using one framework or another may offer different possibilities on the kind of provenance that is captured. For instance, when using Pig Latin to define the transformations it is possible to offer fine-grained how-provenance [33], when using pure MapReduce transformations it is possible to offer fine-grained why-provenance [61], for general MapReduce transformations it may only be possible to offer coarse-grained provenance.

With no pretension of being exhaustive, we present some of the most used frameworks of the Hadoop ecosystem while showing some of their interdependencies, with the purpose of illustrating the challenges it poses for provenance capture and recording. At the core of this ecosystem are the Hadoop Distributed File System (HDFS) [23] and the Hadoop MapReduce [24] frameworks. The HDFS is a distributed file system which is highly fault-tolerant, provides high throughput access to application data and is suitable for applications that have large data sets. Hadoop MapReduce is a software framework for writing applications which process vast amounts of data in-parallel on large clusters in a reliable, fault-tolerant manner. The developer has to solve his problems using the MapReduce programming model where data is accessed as files in the HDFS and the framework will handle parallelism and fault-tolerance.

Regarding the Hadoop Ecosystem, first, there are two very popular frameworks which add a higher-level abstraction to Hadoop, these are Pig [19] and Hive [16]. Both frameworks offer a language to express data transformations, respectively, Pig Latin -a data-flow language- and HiveQL -an SQL dialect-. Transformations declared using these languages are compiled into MapReduce jobs to be executed. [79] There exist solutions which capture provenance for MapReduce jobs (e.g. Ikeda et al. [61]), and since Pig Latin and HiveQL are both compiled to MapReduce jobs, these existing solutions could be used to capture provenance for Pig and Hive too. However, developing a specific solution for Pig or Hive frameworks opens the door to offering a richer notion of provenance by taking advantage of the declarative specification of the transformation provided by their language, as the solution of Amsterdamer et al. [33] which offers how-provenance for Pig Latin transformations.

Besides offering a higher-level abstraction to MapReduce, they both offer an abstraction to the file offered by HDFS. For instance in Pig, the data structures are much richer, typically being multivalued and nested, while Hive organizes data into tables. This should be taken into account when assigning identifiers to the data items, since the notion of data item seen from inside the framework may not match with what that from HDFS. Another important aspect is that when loading a table in Hive (Hive may copy the file into its warehouse directory or refer to the data in its existing location), Hive does not initially modify the file, so a correspondence seems to exist between files and tables. However, Hive offers the possibility of partitioning and sorting tables, which results in physical changes to the files but no changes to the table. [79] It is important to see that a provenance solution which uses positions in a file as identifiers for the data items (e.g. Ikeda et al. [61]) is prone to break if these operations are performed.

Another widely used framework is HBase [15], a column-oriented database built on top of HDFS which offers real-time read/write random-access to very large data sets. The abstraction offered by HBase over HDFS consists in tables, which are multi-dimensional maps, in which a cell is identified by a combination of row key and column qualifier, and contains a value and a timestamp, which represents the value's version. So data items (cells) always have an identifier in

HBase. [54] Since HBase data can be written and read by any application, the challenge here is that of provenance capture.

What is more, HBase's files in HDFS can be used as input or output of MapReduce jobs. Also, HBase tables can be registered as Hive tables [12] and be used as input or output of HiveQL queries as well.

The following example shows the integration of MapReduce, HBase and Hive to illustrate the challenges of offering provenance in this environment. Consider an HBase table which is initially loaded via a MapReduce job MR1, then updated by an application A1 (where some of the data items are used to create other), and then via a HiveQL query HQ1 this table is integrated with a Hive table and recorded in another Hive table. Offering provenance associations in this scenario means being able to associate the input data items of the MapReduce job MR1 with the output data items of HQ1 inside the last Hive table.

To offer provenance in this scenario it is necessary that a specific module be designed for each framework, which is able of capturing the required provenance information inside that framework while associating identifiers to data items and transformations, ideally in a transparent manner and imposing minimal restrictions to the data transformations. Despite these modules being framework-specific, they must work consistently, so the provenance information they produce can be integrated later. This means, for example, that the unit of data produced by MR1 should be consistent with the unit of data accessed by the application A1, and that these data items must be given the same identifiers. If the unit of data produced by the former and consumed by the latter does not coincide, each data item must be assigned a different identifier and the relation between those related but different data items must be recorded. For example, in the example, the records in the result of MR1 may be consider the unit of data and may be given each an identifier, while in the application A1, a record may be seen as a sequence of cells, where a cell is considered the unit of data and each cell is given an identifier. To offer provenance across MR1 and A1, it is vital to be able to relate records produced by MR1 and cells accessed by A1.

There are many other frameworks besides these, but the challenges are the same in general, building framework-specific solutions which are able to capture provenance inside them but still act in a consistent manner. For instance, there exist tools to declare and execute workflows of MapReduce jobs, like Oozie [18], Azkaban [22] and Tez [21]; machine learning tools like Mahout [17]; and in-memory SQL-like frameworks like Impala [25] and Tajo [20].

In figure 4.5 we represent some of the aforementioned systems and the interdependencies between them, using arrows to indicate that the source may invoke and thus depend on the target.

4.2.5 Only inside Hadoop? Centralized or Federated Metadata

If considering data transformations occurring outside of the Hadoop platform, data can thus have a history of transformations both prior being loaded to Hadoop and later when copied from it, and this should be addressed by the solution. Examples of this are data being loaded from the cloud (e.g. Amazon

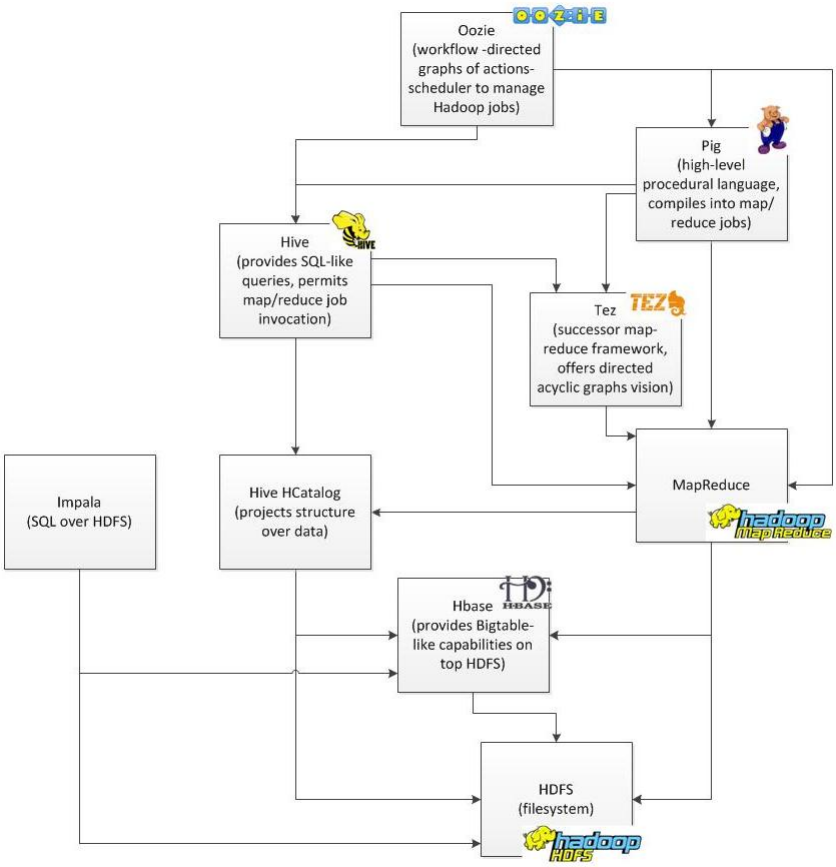

Figure 4.5: A part of the Hadoop ecosystem, representing the different frameworks and the fact that one may invoke and thus depend on others.

S3 [13]) where it had gone through several transformations and thus have provenance in its source system. This data are then loaded (copied) into our Hadoop platform, where several MapReduce transformations (for instance) take place and the result is copied to a DBMS where it is integrated with other data to result in other new data items which are stored in the DBMS (as in our case study).

In this case, it is necessary to design a much more general scheme to capture and manage provenance metadata. First, it is clear that the provenance capture is a system-specific problem but the solution must offer mechanisms to interchange the metadata.

In this distributed scenario the problems of data identification and location arise again with new challenges. How to guarantee that data identifiers are unique across different systems? When data are copied from one system to another, should new identifiers be created for them? How to locate and access a data item given its identifier?

Other aspects of the problem are where provenance metadata should be stored. Should we follow a distributed or a centralized approach? Must provenance metadata be always stored with the data? Different options could be selected for different cases, distinguishing for instance the history prior to the data being loaded in our working space from the history from there on, and distinguishing local data transformation systems from those outside our control.

Regarding data that is loaded into our system, its provenance regarding backward tracing cannot change (the data have been created already). So if we are accessing it from an outside system (and we have no space limitations) we probably want to obtain it together with the data to guarantee its availability and probably faster access to it. In this case, it is necessary to define a common model for the metadata exchange, e.g. OPM or PROV-DM. Alternatively, a query mechanism should be defined to access the metadata on demand.

On the other hand, if we consider only a local environment, we could opt for a centralized metadata repository or a federated one. Since in traditional DBMS contexts fine-grained metadata is reported to be large to be recorded [76], and since in Hadoop the space is abundant and cheap, a good choice seems to be to follow a centralized approach, recording all provenance metadata inside Hadoop. In this case, a write API should be provided to record provenance metadata in Hadoop. Alternatively, should a federated approach be followed, a query mechanism should be defined as mentioned earlier.

In a centralized approach, all provenance metadata would reside in one place, so whenever data transformations are executed by systems outside of Hadoop, these are burdened with the task of sending the metadata to the central repository, either in the act or later in batch mode, which can have an impact on these systems' performance. However, when querying provenance with the centralized approach, the metadata is readily available without need to query the different systems, which can speed up answering.

On the other hand, in a federated approach, systems processing data outside of Hadoop could be requested to report some minimal information, like which data they use, but there is no need for them to report everything that happens which can be a great release. However, these systems will be requested to answer whenever there is provenance querying, so provenance answering will be slower. This problem is exacerbated when there are many systems processing data outside Hadoop and they have many interdependencies, so since no one

has a complete picture of provenance, the systems may be queried iteratively to get the answer.

A solution which can work in a hybrid mode seems to be the best, since it would permit each system running data transformations to select a preferred approach, based on its flexibility, its SLAs, etc.

4.3 A General Framework for Provenance

In this section we develop a solution for the scenario presented in chapter 3. The solution is presented first in a conceptual way and then the details for its implementation are presented in the following chapter.

We summarize the scenario in which we wish to offer data provenance as follows.

- It is a distributed scenario with different systems (e.g. Hadoop, Relational databases, etc.), where data can be moved between them and where data transformations occur in every system.
- Inside the Hadoop environment we expect to run different frameworks to record and transform data.
- We expect to load data from external sources, which could be loaded with their provenance information.
- We require a general definition of provenance, which should be satisfied as good as possible by each system. The requirement for data provenance is to fulfil the conceptual data model developed in section 4.1.
- We are required to address the problem of data identification.
- We prioritize fast provenance answering over provenance capture overhead, thus we opt for a metadata approach.

To support provenance capture and recording in this scenario we present the following framework. The framework defines roles and responsibilities and communication mechanisms. Within this framework it is possible to reuse existing solutions for provenance capture, for example, that of Ikeda et al. [61] for MapReduce jobs and that of Cui et al. [45, 47] for relational queries. We also present a provenance recorder to work within this framework, which is the missing piece to complete the puzzle.

To offer provenance information with a metadata approach, we distinguish two aspects of the problem and hence two basic functions of the solution: first, that of capturing provenance and secondly, that of recording it, which we represent in figure 4.6. They are inherently different functions, and while provenance capture needs to be solved specifically for each system and application context, its recording is a general problem and it suffices to solve it once.

So, in a distributed environment, where different nodes with possibly different technologies perform transformations over data, we need the Provenance Capture function inside each system (e.g. one specific for relational databases, one for Hadoop, etc.). On the other hand, it is not necessary to have a provenance repository in each node, it could be recorded in one central repository if

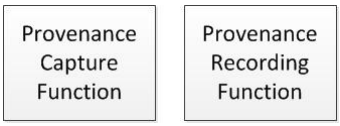

Figure 4.6: Different Functions.

it is considered more convenient (it could though, be recorded in a distributed fashion).

Even if only considering the Hadoop environment, this separation also permits to have different Provenance Capture modules working at the same time, for instance, one for MapReduce jobs and another one for Pig Latin scripts, and yet for both of them to use the same Provenance Recording module.

In figure 4.7 we show a diagram of our case study, where different Provenance Capture functions have been deployed, while only one Provenance Recording function is used. In this example the Provenance Recorder in Hadoop keeps provenance metadata for transformations taking place in the DBMS. Provenance metadata has traditionally been considered too large to be recorded, so Hadoop is the best candidate to manage this big metadata. Therefore we place the Provenance Recorder inside Hadoop.

Provenance information could be recorded in a centralized or distributed manner. In a centralized solution, as in the example in figure 4.7, the Provenance Recording function inside the Hadoop system would record:

- Provenance for data produced inside Hadoop, for transformations run inside Hadoop.
- Provenance for data produced outside of the Hadoop system. Every system which runs a Provenance Capture function records its data provenance using the Provenance Recording function in Hadoop.
- Provenance information to relate data produced in one system and used as input to a transformation in another system, so to be able to trace provenance through different systems.

Alternatively, in a distributed approach to Provenance Recording, every system must run both a Provenance Capture Module function and a Provenance Recording function. Therefore, the Provenance Recording function inside the Hadoop system would record no data provenance regarding data items produced outside of it, since every other system would be responsible of recording it. A consideration must be made, because if every system records provenance solely for its data items, it is not possible to trace provenance when it spans through various systems. To overcome this, when data items are copied from one system to another, the first one must record that those data items were copied to be used by the second system, and when it has to answer a forward tracing provenance query, it knows it must query the second system for that piece of provenance.

When in a distributed environment provenance spans through different systems, in order to answer provenance queries it is necessary to query the different systems sequentially to construct the whole graph with all the ancestral

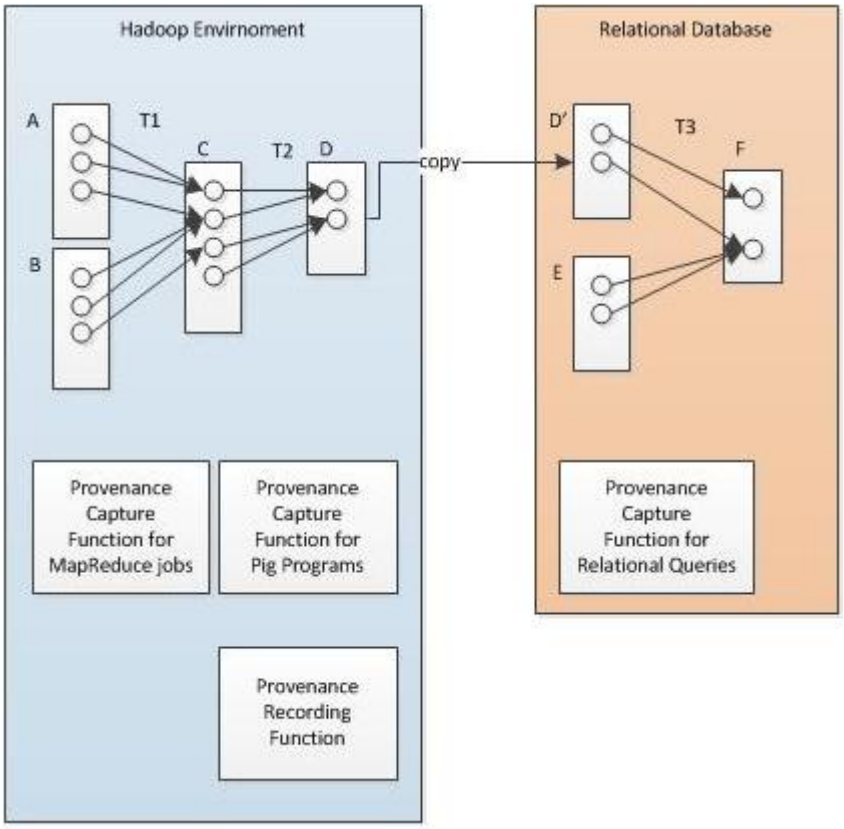

Figure 4.7: Several Provenance capture functions and one Provenance recording function.

and descendant data products. Given the requirement for answering provenance queries fast, we opt for a centralized approach in which all provenance information is readily available in one repository permitting faster access to the information.

What is more, using a centralized repository permits to give the same treatment to all of the metadata created by the different systems in the distributed scenario and thus offer the same levels of: indexes and query response time, fault tolerance, etc.

Another argument in favour of following a centralized approach, more specifically a centralized repository in Hadoop, is that every provenance metadata would be recorded where disk space is considered cheap and there are no restrictions to the amount of data managed. Alternatively, in a distributed approach, metadata is recorded in the different systems which probably have restrictions on the amount of data they manage.

In a metadata approach data transformations' performance is penalized by the overhead of recording the metadata. Yet, in the centralized approach, transformations (and network bandwidth) are further penalized because all the metadata must be sent to the central repository as soon as it is captured, which would take more time than saving it locally and would also use network resources. However, these problems could be mitigated by implementing mechanisms of buffering and programmed synchronization.

Lastly, to be able to use a centralized repository requires that the distributed data transformation systems, which are able to capture provenance, are capable of sending this metadata to the repository in a "push" manner. Besides any efficiency considerations, it may be the case that the system is closed or belongs to a third party and thus cannot work this way. Because of this, the provenance repository must be able to work in a hybrid manner (permitting some metadata to be stored in a federated fashion). That way, when tracing provenance for a data item which was input or output to this closed system, the system is queried for it, thus working as a distributed repository at least for that part of the workflow. In this case it is necessary to know which data items were input or output to that system.

A similar thing happens when one of the nodes in the environment cannot record provenance metadata but still offers a tracing procedure. Analogously, it must also be recorded which data items were input or output to that system, and when tracing provenance for any of those data items, that system should be queried. The difference is that instead of having the metadata readily available it needs to compute it.

The Provenance Recorder function we propose can work in both ways, being able to record for some data items their complete forward tracing provenance for if that is the information it receives, and for other data items recording only which systems are using them.

4.3.1 Provenance Capture Function

The Provenance Capture function, which is system specific, is responsible of capturing provenance metadata and sending it to the Provenance Recorder. Whether we have a centralized or a federated repository, the capture function is the same. In order to do this, first it must address the following problems:

1. Assign each data item an identifier.
2. Recognize which transformation is being executed, register the context of the execution (e.g. who ran it, when, using which parameters, etc.) and assign an identifier to the transformation execution.
3. Capture provenance for the created data items.

We analyse each of the three problems and present the proposed solution.

Identifying Data Items

In order to record provenance metadata, i.e. which data items were used to create some others, it is necessary to have a means of referring to those data items. There are two possible ways to do it, one is to use the data items' values while the other one is to assign each data item an identifier which identifies it univocally. We consider it necessary to use identifiers because:

- Using an identifier gives an univocal way to identify data items, specially when data sets may have duplicated elements. Although two data items in the same data set may have the same value, they are still not the same data item. These can have different creation time and therefore could have been used as input by different transformations ran in different moments.
- Also, using identifiers to refer to a data instead of the data item's value, makes the reference a uniform (data items that may reside in very different systems and have different formats) and compact one.

Therefore, the identifiers and not the actual data items are sent to the Provenance Recorder, so the communication and recording is done in a uniform and more compact manner.

Once established the need for data items' identifiers, there are three things to consider:

- The identification of data items and resolving their location are two different problems.
- Assigning identifiers to data items is a system specific problem.
- For a general solution, which integrates data provenance from different systems a uniform identifier is desirable.

Identification vs. Location The location of a data set is unique, whether it is a URL or a path, following it will lead to one and only one data set. Analogously, the physical location of a data item in that data set is also unique inside the data set. If the two are joined, it results in a global unique location for a data item. It may be tempting to use this data item location as an identifier, but this results in a brittle solution, as has been shown already. Whenever the data set is moved or its data items are rearranged, the identifiers must be updated, which shows they were not really identifiers. It would be really inconvenient having to update the data provenance repository because a file has changed location.

Assigning identifiers to data items is a system specific problem Every system has its own characteristics and thus has different possibilities regarding the creation and assignment of identifiers to data items. In a distributed environment, where different systems are running data transformations and data provenance must span through all of them, it is necessary -as was just shown- to ask these systems to provide identifiers for the data items, but it is not necessary, and it is even inconvenient to impose them a data identifying scheme.

For example, in a relational database, a relation may have a primary key, in which case identifying data items could be considered a trivial matter. Alternatively, if the relation does not have a primary key another mechanism must be used. Whatever the way data items are identified, it should be transparent to the Provenance Recording function, that only deals with data identifiers.

Each system should develop the data item identifying scheme that better suits it. What is more, each system could develop all the identifying schemes it needs. What is needed from the system is for it to offer a layer of abstraction on how it is solving the problem, to deliver provenance associations using data identifiers, and once a data item is assigned an identifier, the system is able to maintain the identifier-location association and thus to be able to return a data item when given its identifier.

Global identifiers Despite what was just said about of the assignment of identifiers being a system specificity, a data provenance solution which spans through various systems requires those identifiers to be uniform and globally unique, for it to be able manage them.

To achieve global uniqueness of identifiers we propose to follow a hierarchical distributed id assigning scheme, similar to the Internet Domain Name System.

An identifier for a data set consists of two numbers separated by dots ('.').

$$\textit{'system'} \,.\, \textit{'data set'}$$

A data item's identifier consists thus of three numbers separated by dots ('.').

$$\textit{'system'} \,.\, \textit{'data set'} \,.\, \textit{'data item'}$$

The first number corresponds to the systems which runs the data transformation that creates the data item and that records it. The second number corresponds to the data set where the data items is recorded. This number should be unique in the context of the system. The third number corresponds to the data item itself, which should be unique in the context of the data set.

Therefore, uniqueness of identifiers for data sets and data items is guaranteed because:

- A central repository assigns a unique identifier to each system running data transformations.
- Each system runs a Data set catalogue and assigns a unique identifier to each data set.
- The Provenance Capture module running in each system assigns -using the identification scheme- each data item an identifier that is unique in that data set.

Since we record provenance for the data items' identifiers, it is therefore necessary to locate the data items when provenance is queried. Hence it is necessary to maintain the association between the identification and the location of data items. Since data sets (and thus data items) can be moved, the best suited to be informed about these changes is the system where the data resides. We create a module called Data Set Catalogue running in each system, so it should take responsibility of maintaining this information. We assume that data cannot be deleted.

The Id Creation Scheme module participates assigning identifiers to data items as they are created, but also when provenance is queried and data items must be retrieved by their identifiers, this module is responsible for retrieving the data items.

The Id Creation Scheme module works very close to the Provenance Capture module, because when the latter captures the provenance association between data items, it must ask the former for their identifiers.

Also, the Id Creation Scheme and the Data Set Catalogue modules are very interrelated, because should the identification scheme be based on some physical characteristic of the data set, if the data set was modified the Data Set Catalogue should notify the Id Creation Scheme module. An example of this is when data items are identified by their physical position in their data set. In this case, the Id Creation Scheme module should maintain the association between identifiers and locations. If the data set was rearranged (ordered for example), the Id Creation Scheme should update the identifiers-locations association.

We point out that when copying data between systems, it is vital that it is copied with its identifiers. An alternative is to assign them new identifiers and to declare the relation between them, as the identity transformation. The latter option is recommended when the second system cannot manage the identifying scheme or the data granularity changes, so new identifiers are actually needed.

Identifying Transformation Executions

Another system specific problem is identifying the transformation being executed, the execution itself and the context of the execution.

Identifying the transformation being executed requires the existence of a Catalogue of transformations, ideally containing the author and a description, as well as other required information. This catalogue could allow the identification of different versions of the transformation, and therefore associating the transformation executions and the data items created to the corresponding version of the transformation.

The context of the execution would include aspects such as: who executed it, parameters passed to the transformation and what was the system state at the moment.

In order to register transformation executions, each execution must be assigned an identifier as well. An identifier for a data transformation consists thus of two numbers separated by dots ('.').

'system' . 'execution number'

The first number corresponds to the system which runs the data transformation while the second number corresponds to the execution. Each system is

responsible of assigning each execution an identifier which is unique within its domain.

When a data transformation is executed, the system running it should register the following information:

<transformation execution id, transformation specification id, execution context >

Regarding the identification of the specifications of data transformations, two approaches are possible. One is for each system to have its local Transformation Catalogue being responsible for assigning identifiers to them. This has the inconvenience that should two identical data transformations be available in two different systems they would be assigned two identifiers. On the other hand, if a centralized Transformation Catalogue responsible for doing it, data transformations could be registered only once and they could be referenced from local Transformation Catalogues that would contain only the available transformations in that system.

In either case, the local Transformation Catalogue running inside each system is necessary and is responsible for maintaining the relation between the specification of data transformations and their executable counterparts. There is also a need for a centralized Transformation Catalogue which knows all the existing data transformations and is able to declare workflows which span through different systems.

When running workflows we face very similar problems. The workflow execution corresponds to a workflow specification, which must also be registered as follows.

<workflow execution id, workflow specification id, execution context >

A workflow specification is composed of a series of data transformation specifications which should be executed following a certain specified logic. When a workflow runs, it will follow that specified logic to invoke the different data transformations. When it calls the data transformations it is informed of their identifiers which it must register as follows:

<workflow execution id, transformation execution id, position in workflow >

In figure 4.8 we extend the diagram of the case study to show the relationships between workflow and data transformation executions and their specifications inside the Transformation Catalogue.

Provenance Capture

When a data transformation runs, consuming data and producing data, the Provenance Capture module is responsible for identifying the relation between input and output and record it.

Using the identifiers provided the Id Creation Scheme and the Data Set Catalogue for the input and output, and the identifier assigned to the transformation execution, the Provenance Capture module will record both coarse- and fine-grained provenance respectively as follows:

<output data set id, [input data set id], transformation execution id >

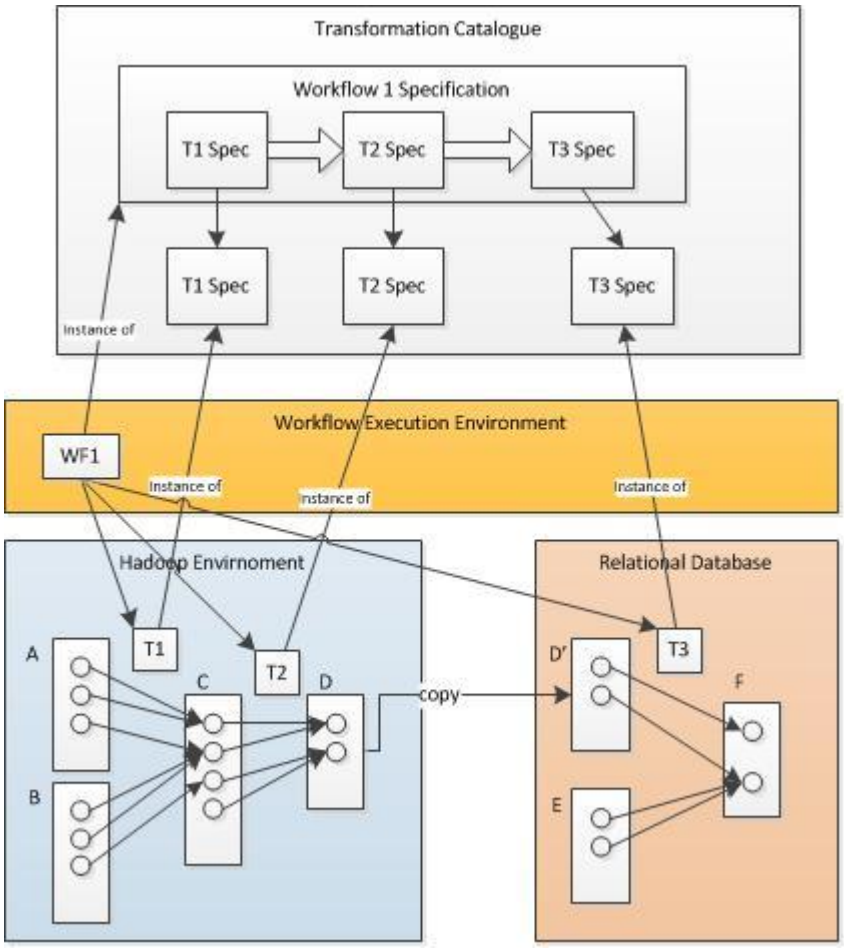

Figure 4.8: Three data transformations (T1, T2 and T3) being invoked by a workflow WF1. The arrows between WF1 and T1, T2 and T3 represent WF1 invoking them.

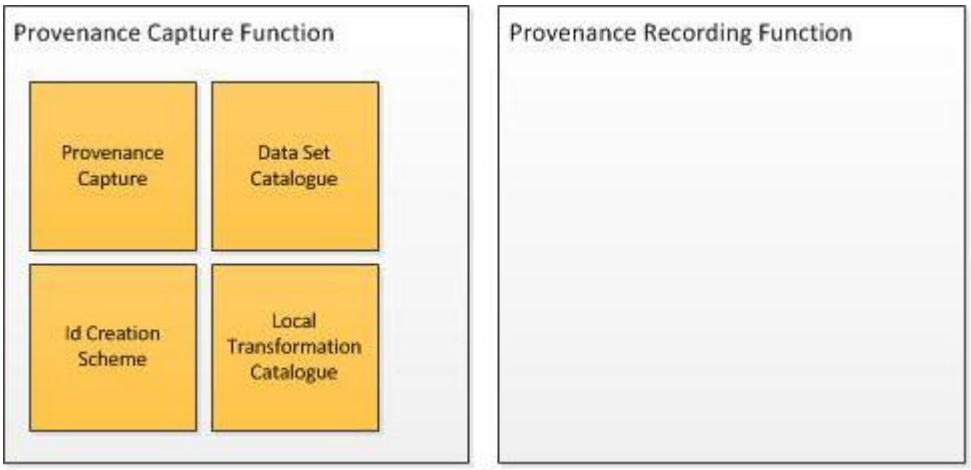

Figure 4.9: Fundamental aspects of the provenance capture function.

<output data item id, [input data item id], provenance semantics, transformation execution id >

These declarations state that the output element (a data set or data item) was created using the (possibly many) input elements by a certain transformation execution. Regarding fine-grained provenance, should it be possible to capture the semantics of the relation between input and output data items (e.g. why-p, how-p or other to be defined), it could be recorded as well.

We point out that it may be possible for a system to offer different Provenance Capture modules, one to address each system specific characteristics. For example, it could be possible in a Hadoop environment to develop a Provenance Capture module specific for MapReduce jobs and another one for Pig programs. The latter would be able to offer more complete provenance information than the former, yet it can only be applied when running a Pig programs. Analogously, in a relational database two different Provenance Capture modules could be implemented for relational queries and for ETL programs.

If several Provenance Capture modules were implemented, they should be served by the same Id Creation Scheme and Data set Catalogue, for it to be possible to trace provenance consistently through these different types of transformations.

Provenance Capture Function Summary

The Provenance Capture Function is thus composed of the following modules, as represented in figure 4.9: Provenance Capture, Data Item identification Scheme, Data Set Catalogue and Data Transformation Catalogue.

When a data transformation is executed, the corresponding Provenance Capture module (there may more than one inside one system, each for some specific type of data transformation) runs and captures the provenance association between input and output data items of input and output data sets. To do this, the Provenance Capture module requires:

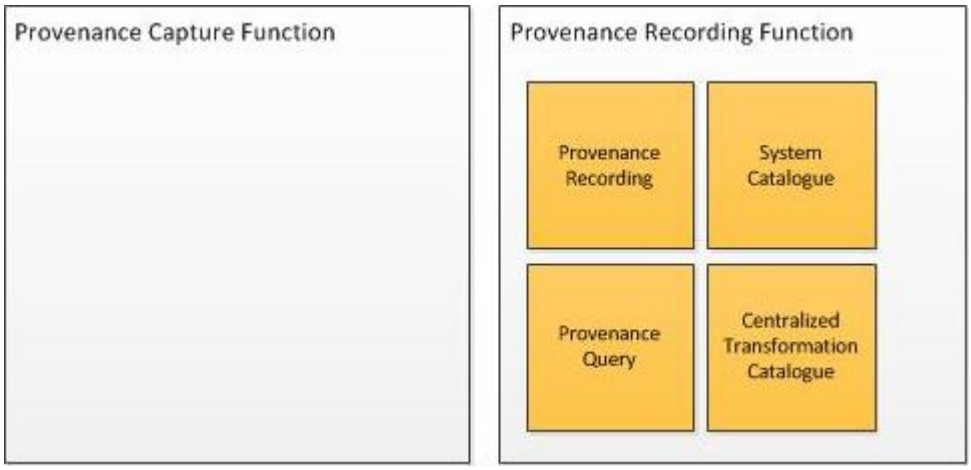

Figure 4.10: Fundamental aspects of the provenance recording function.

- the Id Creation Scheme and the Data Set Catalogue to provide the identifiers for the input and output data items.
- the Transformation Catalogue to provide the identifier of the transformation execution.

Then, all this information is sent to the Provenance Recording Function (for it to record it).

4.3.2 Provenance Recording Function

The Provenance Recorder function, represented in figure 4.10, consists of: the System Catalogue module, which maintains the list of data transformation systems for which provenance is recorded; the centralized Transformation Catalogue; the Provenance Recorder module, which will receive all the provenance metadata and record it; and the Provenance Query modules, which translate the provenance metadata to the required provenance models.

Systems Catalogue

First, there is the Systems Catalogue, which is responsible for registering all the systems running data transformations in the distributed environment and assigning each one an identifier. This identifier is used by each one as part of their id creation schemes to be able to create identifiers which are unique through the whole distributed environment.

The Systems Catalogue also maintains the list of the nodes recording provenance in the distributed environment and the information to communicate with them.

When querying provenance metadata and obtaining data items' identifiers, the first level part of the identifiers is used to know which system's Data Catalogue to query for those data items. The Systems Catalogue provides the information needed to query the Data Catalogue .

Centralized Transformations Catalogue

As was shown in the previous section, there is a need for a centralized Transformation Catalogue which knows all the existing data transformations and is able to declare workflows which span through different systems.

Provenance Recording

The Provenance Recorder module receives all the provenance metadata that was described in the previous section and has responsibility for recording it.

Summarizing the previous section, we list the metadata information that is recorded:

- When a workflow is executed it registers:

  <workflow execution id, workflow specification id, execution context >

- When a workflow runs and invokes a data transformation it register:

  <workflow execution id, transformation execution id, position in workflow>

- When a data transformation is executed it registers:

  <transformation execution id, transformation specification id, execution context >

  <output data set id, [input data set id], transformation execution id >

  <output data item id, [input data item id], provenance semantics, transformation execution id >

As mentioned earlier, it is possible that some of the systems running data transformations are not able to send the provenance metadata to the Provenance Recorder, but still offer some support for provenance (be it by metadata or a tracing procedure). In this case the Provenance Recording function works as a federated repository.

To perform backward tracing over data items produced by these systems, it suffices to know that they produced the data in question. The information necessary to know this is already available if the system is registered in the System Catalogue and the data identification follows the specified format. To perform forward tracing over the data items used by these systems, it is necessary to know which data items were consumed by them. For this we incorporate the possibility for the system to declare which data it consumes:

<system id, input data set id >

Defining this communication interface for the different nodes to send the provenance metadata permits to offer different implementations for its physical recording without modifying the recording functions. For example, the provenance metadata could be physically recorded in different ways, favouring backward or forward tracing operations or both depending on the access requirements and space availability.

Also, additional data structures could be incorporated to facilitate provenance tracing in a sequence of transformations, when data items which are the result of a data transformation are then used as input for another transformation, so provenance tracing requires an iteration over the provenance metadata.

Provenance Query Module

Should there be a requirement for exposing the provenance metadata using a certain data model, the translation from the internal representation to the desired representation would be done by the Provenance query module.

4.4 Conclusions

In this chapter we have developed a framework for capturing and recording data provenance which satisfies the requirements of the case study. For this, we have first developed PCM, a conceptual data model for provenance, based on the data models of the surveyed solutions. We have also presented an analysis of the general aspects of the provenance problem as well as of the particular difficulties that arise in the case study.

The main contribution in this chapter is the framework we have presented, defining roles and assigning them responsibilities. Our framework permits to capture and record provenance through different systems with different technologies (in particular different frameworks of the Hadoop ecosystem) within a distributed environment, a thing not offered by any of the works we have reviewed.

Our solution is flexible since it allows participating nodes in the environment to send provenance metadata to the repository -the default option- or alternatively work in a federated fashion, where they can record provenance locally or compute it via a tracing procedure.

Chapter 5

Implementing the Framework

We have presented a framework for capturing and recording data provenance in a distributed environment, defining roles and assigning them responsibilities. Our framework is based on a centralized metadata repository, where provenance capture occurs in different nodes in the distributed environment and is sent to the metadata repository. We propose to reuse some of the reviewed solutions which have already solved provenance capture in different contexts and design the provenance recording module, the piece which completes the puzzle. This metadata repository offers services to the provenance capture nodes for them to record the metadata.

First, in order to define the communication between the provenance capture modules and the provenance recording module, we survey existing models for provenance interoperability, select two (OPM and PROV-DM) and assess them by comparing them to PCM, and decide to use PROV-DM to define the operations, in order to obtain a more general solution.

In the next section we define the services offered by our metadata repository (based on the operations required in the case study) and define a concrete recording strategy using Hadoop and HBase (a column-oriented database built on top of HDFS which offers real-time read/write random-access).

In section 3 we present general aspects of the implementation of provenance. Then we review the surveyed works to see the details of their implementations and close the section with a summary of the reviewed solutions. Finally, in the two final sections we show how the reviewed solutions of Ikeda et al. for MapReduce jobs and that of Cui et al. for relational databases, can be adapted to work within the proposed framework.

5.1 Interoperability Provenance Data Models

Several models to represent provenance for communication and interoperability have been proposed, yet only few of them have been adopted. For instance, the Provenir ontology, which was designed for use in e-science in 2009 has received few citations [75] (cited by 25 according to Google Scholar, September 5th 2015) and the project's home [39] page has not been modified since May 2011 (last checked September 5th 2015). The Provenance Vocabulary [57] was designed by for data on the web and to deal with the trustworthiness of RDF

data in 2009 and since then it has received 152 citations (according to Google Scholar, September 5th 2015). However, its home page [58] does not seem very active, being last updated on July 2013 (last checked September 5th 2015). The Provenance Markup Language (PML) [66] (originally the Proof Markup Language [48] was designed for sharing explanations generated by various automated systems such web agents. The project's home page [50] states that version 3.0 is in process of being designed, yet it was last modified on February 2013 (last checked September 5th 2015). The Semantic Web Applications in Neuromedicine (SWAN) Ontology [43] is an ontology proposed in 2008 for modeling scientific discourse to facilitate the exchange of biomedical information in context. Its page in Google Code [42] has been archived (last checked September 5th 2015).

Simultaneously (2006-2010), the Provenance Challenge workshops [4] took place, which resulted in the Open Provenance Model (OPM) [67, 29]. The Fourth and Last Provenance Challenge [68] in 2010 resulted in the constitution of the Provenance Incubator Group [7], where this community of researchers continued their work. Using OPM as a reference model, the W3C Provenance Incubator Group defined mappings to existing vocabularies and models, including the aforementioned ones [56]. Later this group was renamed the W3C Provenance Working Group [69] and, based on OPM, developed the PROV Data Model (PROV-DM) [9] which is a W3C recommendation since April 2013.

It is worth noting that many of those researchers participating in the aforementioned models and projects took part in the development of the W3C PROV-DM, which would explain why the projects were discontinued.

The Dublin Core Metadata Initiative (DCMI) [5] provides a widely used core metadata vocabulary (commonly referred to as Dublin Core) for simple and generic resource descriptions, which, although is focused on describing resources in a general sense, includes terms to provide information related to the provenance of the resource. The original Dublin Core Metadata Element Set (a.k.a. legacy) namespace URI `http://purl.org/dc/elements/1.1/` has limitations such as the properties have no specified ranges, meaning that arbitrary values can be used as objects. However, the terms namespace (a.k.a. DC Terms) URI `http://purl.org/dc/terms/`, replicates the fifteen properties from the original namespace and includes additional properties and classes, permitting the definition of ranges for the properties.

The Provenance Incubator Group has defined a mapping between the DC Terms and OPM [56], and between DC Terms and the PROV Ontology (PROV-O) [8].

Today, the most widely used models are PROV-DM and OPM, which are also recommended for provenance communication in A Primer on Provenance [38] in ACM Queue. The mainstream workflow systems today, Kepler and Taverna, though they have their own internal representation, permit exporting provenance: Kepler to OPM [28, 29] and Taverna to OPM and PROV-DM [27, 26].

The objective of both OPM and PROV-DM is not to be adopted for internal representation for existing systems, but instead to be used for serializing internal provenance representations in order to allow existing systems to communicate and interoperate [6]. Both OPM and PROV-DM are serializable into XML [3] and [11] respectively and offer an OWL Ontology model, OPMO [1] and PROV-O[10] respectively. However, the available specifications for the OPMO (OPM

OWL Ontology) and OPM XML schema are working drafts, both of October 2010 (last checked September 5th 2015).

Next we review the Open Provenance Model and PROV-DM and analyse how fit they are to represent the provenance definitions aforementioned.

5.1.1 Open Provenance Model (OPM)

Moreau et al. [67] have developed a model of provenance, called Open Provenance Model (OPM), as the result of the Provenance Challenge series. This is not a system that captures or records provenance, but only a model. The model is designed to allow the definition of provenance for anything (digital or not) in a precise, technology-agnostic manner and hence allow provenance information to be exchanged between systems. The OPM can be used to model relationships between artifacts, and this can be used to compute provenance (forward-tracing and backward-tracing, namely why-provenance).

The OPM does not specify nor restrict how systems should internally represent, manipulate and store their metadata, its focus is on information exchange between systems. Serializations in XML and OWL are available in the OPM website [29].

The model is very general, dealing with the creation of Artifacts, which are defined as "immutable pieces of state, which may have a physical embodiment in a physical object or a digital representation in a computer system". Artifacts are created by Processes, i.e. "actions or series of actions performed on or caused by artifacts, and resulting in new artifacts". Finally, an Agent is a "contextual entity acting as a catalyst of a process, enabling, facilitating, controlling, or affecting its execution".

Regarding streams of data, the authors consider an artifact to be a slice of stream in time, i.e. the stream content at a specific instant in the computation.

Over the mentioned elements, a causal relationship is represented by an arc and denotes the presence of a causal dependency between the source of the arc (the effect) and the destination of the arc (the cause). Five causal relationships are recognized, which we represent graphically in figure 5.1:

- A process used an artifact, identifying under which role the artifact was used.

- An artifact was generated by a process, identifying under which role the artifact was generated.

- A process was triggered by a process, expresses a necessary condition, the latter was required to have started for the former to be able to complete. The authors point out that this interpretation is "weaker than the common sense definition of trigger, which tends to express a sufficient condition for an event to take place". Indeed it is much weaker.

- An artifact A2 was derived from an artifact A1, indicates that artifact A1 needs to have been generated for A2 to be generated. The piece of state associated with A2 is dependent on the presence of A1 or on the piece of state associated with A1

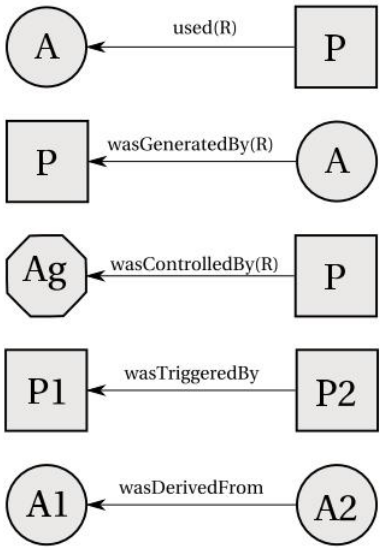

Figure 5.1: Edges in the Open Provenance Model: sources are effects, and destinations causes [67].

- A process was controlled by an agent, since a process may have been controlled by several agents, it is also possible to identify their roles as controllers.

The fact that a process used an artifact and generated another does not imply the latter was derived from the former; such relationship needs to be asserted explicitly using the "Artifact Derived from Artifact" edge.

Regarding the roles of artifacts used by or generated by processes the authors consider they are meaningful only in the context of the process where they are defined. Therefore, the meaning of roles is not defined by OPM and OPM only uses roles syntactically (as "tags").

The Open Provenance Model allows for causality graphs to be decorated with time information. This is not intended to be used for deriving causality. Time may be associated to "instantaneous occurrences in a process", which are for artifacts, the occurrences of creation and use, and for processes, their starting and ending.

Other objectives of the model are to allow multiple levels of provenance description to coexist and to define a core set of rules that identify the valid inferences that can be made on provenance representation. Regarding the first one, given a provenance graph, two sub-graphs can offer different levels of explanation for the same execution (the most detailed one is said to be a refinement of the other). This is not to be confused with coarse and fine granularity in provenance. It is possible to use refinements repeatedly to create a hierarchy of accounts.

There is a limitation with the overlapping accounts though, since it is not possible to declare the relationship between a process and its refinement pro-

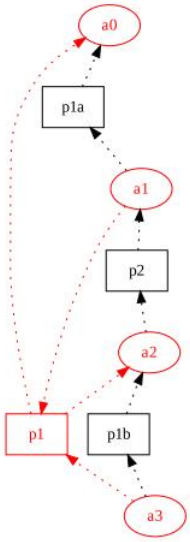

Figure 5.2: Example of Hierarchy of Accounts [67].

cesses. This is made clear in the example of the figure 5.2, where the authors claim that the red account shows that the processes p1a and p1b constitute a single process p1. This is not said explicitly but implicitly by stating that p1a and p1b share their input and output with p1.

Provenance graphs are aimed at representing causality graphs explaining how processes and artifacts came out to be and can be summarized by means of transitive closure. When users want to find out the causes of an artifact or a process, they may not just be interested in direct causes, but in indirect causes, as well, involving multiple transitions, which can be obtained by the transitive closure of the "was derived from" edge.

The authors define a transitive closure over the provenance graphs to summarize the causal dependencies. For this, they introduce completion rules and define multi-step inferences.

First, they define three completion rules: The first one, known as artifact elimination rule, states that a "was triggered by" edge can be obtained from the existence of "used" and "was generated by" edges. Second, the artifact introduction rule allows to establish that the "was triggered by" edge is hiding the existence of some artifact used by P2 and generated by P1. These are represented in figure 5.3.

We point out that this second rule excludes the possibility that the "was Triggered By" edge be used to declare explicitly that a process composes another one, which is the idea of a workflow, or a multiple level description. Should a process P1 be composed of processes P1a and P1b in that order, it would not be possible to declare that P1a "was Triggered By" P1, since that would imply there exists an Artifact "Generated By" P1 which was "used" by P1a, which is not necessarily true.

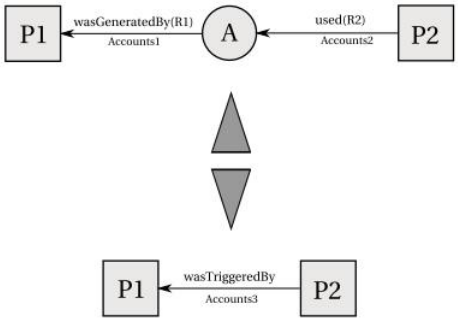

Figure 5.3: Completion rule for Process Introduction. [67]

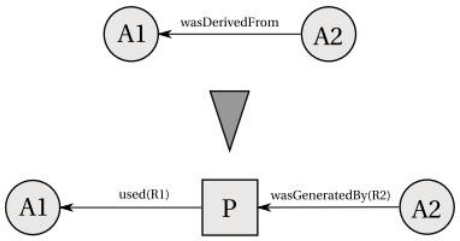

Figure 5.4: Completion rules for Artifact Introduction and Elimination [67].

The third rule, known as process introduction, states that a "was derived from" edge between artifacts hides the presence of an intermediary process. It is represented in figure 5.4.

The authors point out that the completion rules allow us to establish the existence of some artifact but it does not tell us what its id is.

The multi-step inferences are used to derive new multi-step versions of the aforementioned relationships, in order to express that artifacts or processes were related not only in direct causes but also in indirect causes, through multiple transitions.

Four multi-step relations are defined:

- The Multi-Step WasDerivedFrom is defined as the transitive closure of the edge "was derived from", where an artifact a1 was derived from a2 if a1 "was derived from" an artifact that was a2 or that was itself derived from a2 (possibly using multiple steps).

- In the Multi-Step Used relation, a process p is said to have used an artifact a if p used an artifact that was a or was derived from a (possibly using multiple steps). In this inference the Artifact Introduction Rule may be vital to infer that a process p was derived from an artifact a if the only thing known is that p wasTriggeredBy another process.

- An artifact a was generated by process p (possibly using multiple steps), if a was an artifact or was derived from an artifact (possibly using multiple steps) that was generated by p.
- A process p1 was triggered by process p2 (possibly using multiple steps), if p1 used an artifact that was generated or was derived from an artifact (possibly using multiple steps) that was itself generated by p2.

The identification of Artifacts is not addressed by the authors neither in the conceptual documents nor in those regarding serialization. For serialization there are two possible options: XML and RDF. When using XML [3], each artifact [2] is required to have an XSD ID attribute [30] which uniquely identifies an element in an XML document (not necessarily a globally unique identifier). This identifier is used to refer to the artifact from inside the document, for example, to say the artifact was used or produced by a process. The artifact can also have: a label

5.1.2 OPM to represent PCM

A correspondence could be established between our conceptual model PCM and OPM as follows: Data Collections and Data Items could be represented by OPM Artifacts; Data Transformation Executions and Workflow Executions could be represented by OPM Processes; and Data Transformation Specifications and Workflow Specifications could be represented by OPM Agents.

Provenance granularity is not addressed by OPM, that is why we map OPM Artifacts to both Data Collections and Data Items. This is a limitation of the model, since it does not distinguish entities that are conceptually different, and if both were represented as OPM Artifacts it would not be possible to declare the relation between them.

A similar thing happens with workflows. Workflows are not addressed directly by the model, but it could be represented using the multiple levels of description offered by the model. Also, it would be consistent with the definition of Artifact, which is "an action or series of actions". However, using this model it is not possible to declare the relation between a process and its processes in the refinement explanation. We point out that the "wasTriggeredBy" relation offered by the model cannot be used for this, since it implies -according to the specification- the existence of artifacts generated by the former and used by the latter, which would not be necessarily true.

The stronger limitation though is related to specifying the semantics of the derivation between Artifacts, for instance to express how-provenance. OPM permits for roles to be attached to the relations between Artifacts and Processes (i.e. the "used" and "wasGeneratedBy" relations), but it does not permit to define a role to the relation "wasDerivedFrom" between Artifacts. Specifying the role a data item plays as input to a data transformation is not enough to know the role it played for each output data item.

5.1.3 PROV-DM

PROV-DM [70] is the conceptual data model that forms a basis for the W3C provenance (PROV) family of specifications, a W3C Recommendation since April 2013. The authors define provenance as "a record that describes the

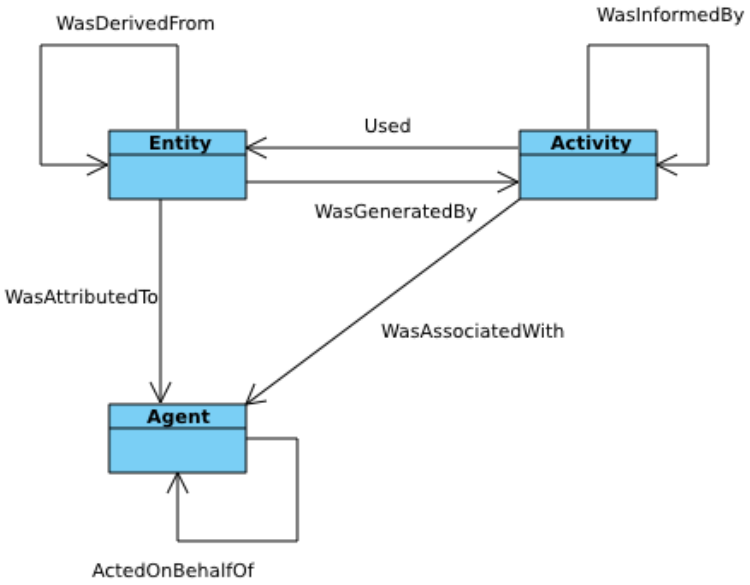

Figure 5.5: PROV Core Structures [70].

people, institutions, entities, and activities involved in producing, influencing, or delivering a piece of data or a thing".

PROV-DM distinguishes core structures from extended structures:

- Core structures "form the essence of provenance information, and are commonly found in various domain-specific vocabularies that deal with provenance".
- Extended structures "enhance and refine core structures with more expressive capabilities to cater for more advanced uses of provenance".

With its core structures, PROV-DM intends to describe the use and production of entities by activities, which may be influenced in various ways by agents. These core types and their relationships are illustrated by the UML diagram of figure 5.5. This core model is based on OPM, since it includes all its concepts and relations with the same meaning (though renaming some of them).

An entity is defined as "a physical, digital, conceptual or other kind of thing with some fixed aspects; entities may be real or imaginary". An activity is "something that occurs over a period of time and acts upon or with entities; it may include consuming, processing, transforming, modifying, relocating, using, or generating entities". An agent is "something that bears some form of responsibility for an activity taking place, for the existence of an entity, or for another agent's activity". According to the specification, an agent could be a particular type of entity or activity, so the model can be used to express provenance of the agents themselves.

Activities and Entities are associated with each other in two different ways: activities use entities and activities generate entities. Generation is "the completion of production of a new entity by an activity. This entity did not exist before generation and becomes available for usage after this generation." Usage

is "the beginning of utilizing an entity by an activity. Before usage, the activity had not begun to utilize this entity and could not have been affected by the entity."

They refer to the generation of an entity by an activity and its subsequent usage by another activity as communication, represented by the association "wasInformedBy". Communication is the exchange of some unspecified entity by two activities, one activity using some entity generated by the other.

A derivation ("wasDerivedFrom" association) is "a transformation of an entity into another, an update of an entity resulting in a new one, or the construction of a new entity based on a pre-existing entity". Attribution ("wasAttributedTo" association) is "the ascribing of an entity to an agent". An activity association ("wasAssociatedWith" association) is "an assignment of responsibility to an agent for an activity, indicating that the agent had a role in the activity".

Finally, delegation is "the assignment of authority and responsibility to an agent (by itself or by another agent) to carry out a specific activity as a delegate or representative, while the agent it acts on behalf of retains some responsibility for the outcome of the delegated work" and is represented by the "actedOnBehalfOf" association.

Regarding its Extended Structures, the authors first define the mechanisms to do it, and then use them to present the extensions. There are four mechanisms defined to extend the model: 1) subtyping core types and core relations (e.g. a software agent is special kind of agent; a revision is a special kind of derivation), 2) expanding core binary relations to to n-ary relations (then the binary relation is seen as a shorthand that can be 'opened up'), 3) optional identification for relations (for when it is required to identify an instance of an association between two or more elements), and 4) new relations.

Based on the Core Structures and applying the defined mechanisms, the following Extended Structures are defined:

- A Software agent is running software.
- An organization is a social or legal institution such as a company, society, etc.
- A Person refers to people. The last three concepts are obtained by subtyping the core structure agent.
- A Revision is a special kind of derivation, defined as a derivation for which the resulting entity is a revised version of some original.
- A Plan is defined by subtyping the core structure entity and full association by an expanded relation, as follows.
  - A plan is an entity that represents a set of actions or steps intended by one or more agents to achieve some goals.
  - An activity association is an assignment of responsibility to an agent for an activity, indicating that the agent had a role in the activity. The expanded relation allows for a plan to be specified, which is the plan intended by the agent to achieve some goals in the context of this activity.

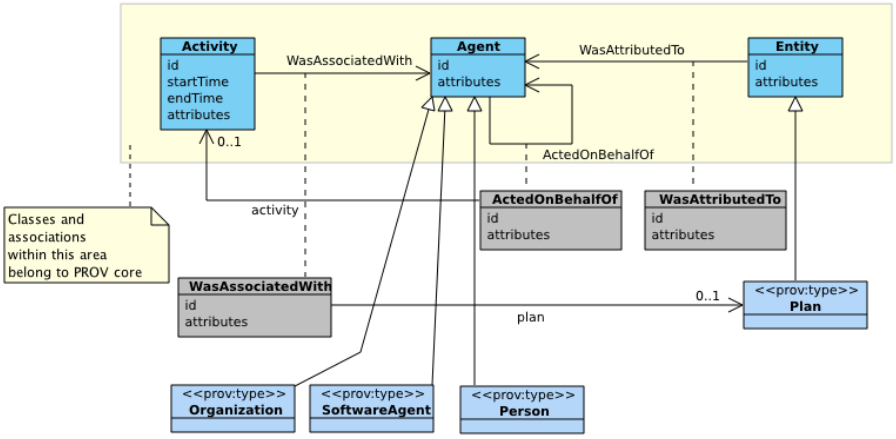

Figure 5.6: Agents and Responsibility Overview [70].

The rationale for considering plans as entities is that since they may evolve over time, it may become necessary to track their provenance. In figure 5.6 we present an overview of Agents and responsibility.

An example of association between an activity and an agent involving a plan is: an XSLT transform (an activity) launched by a user (an agent) based on an XSL style sheet (a plan).

- A bundle is a named set of provenance descriptions, and is itself an entity, so allowing provenance of provenance to be expressed. This is created by subtyping Entity. This is useful for users to analyse the provenance of provenance information, to determine the agent its provenance is attributed to, and when it was generated.

- A collection is an entity that provides a structure to some constituents that must themselves be entities, which are said to be members of the collections. The Collection is defined by subtyping Entity and creating a relation between Entity and Collection. The model permits thus to express the provenance of the collection itself in addition to that of the members.

5.1.4 PROV-DM to represent PCM

A correspondence could be established between our conceptual model PCM and PROV-DM as follows: Data Collections and Data Items could be represented by PROV-DM Entities, in particular, Data Collections could be represented by PROV-DM Collections and use the HadMember relation to declare its members.

Data Transformation Executions and Workflow Executions could be represented by PROV-DM Activities; while the relation between them (the triggering) can be represented by the PROV-DM relations wasStartedBy and wasEndedBy. These PROV-DM Activities would also be related via the the wasInformedBy relation, since it is assumed there is a special Entity called Trigger that is exchanged between them.

Finally, Data Transformation Specifications and Workflow Specifications could be represented by PROV-DM Plans, which are a kind of Entity. There is no way to declare the relation between the two. This could be achieved by subtyping Plan, analogously to Collections, to include Plans composed of Plans.

The Concepts of Roles played by both Data Collections and Data Items in the Transformation Executions can be represented by the Extended versions of PROV-DM Used and WasGeneratedBy, which relate Activities to their input and output Entities and permitting for the definition of additional attributes.

The concept of Data Production, which explains how a Data Item was produced could be represented by the Extended version of PROV-DM WasDerivedFrom, which explains how it was derived, identifies the responsible Activity and permits for the definition of additional attributes.

No correspondence to PROV-DM Agents since in the comprehensive model there is no entity to represent the one responsible for a data transformation or workflow execution.

5.1.5 Summary

We have reviewed the existing provenance models for provenance interchange, which showed the most widely used models are OPM and PROV-DM. The objective of both OPM and PROV-DM is not to be adopted for internal representation for existing systems, but instead to be used for serializing internal provenance representations in order to allow existing systems to communicate and interoperate. Both OPM and PROV-DM are serializable into XML and offer an OWL Ontology model.

We analysed how fit both models are to represent the provenance model built in section 4.1. We found that there were limitations of the OPM model that restricted its possibility of representing several elements of our provenance model.

PROV-DM is an extension of OPM, including OPM's model as its Core Structures, and incorporating Extended Structures, i.e. additional Concepts and Relations. Using these, it is possible to lift the restrictions imposed by OPM, so it is possible to represent almost everything of our model. We could not represent the Specification of Workflows using the entities of PROV-DM, however, it can be represented using the mechanisms to define extended structures offered by the specification.

OPM and PROV-DM are the two most widely used models nowadays. From them, PROV-DM includes OPM and extends it, resulting in a richer model which could satisfy all the requirements set by our comprehensive model. What is more, PROV-DM is a W3C recommendation. Therefore, we suggest its use for interoperability.

5.2 Provenance Recording Repository

In this section we define the services offered by our metadata repository (based on the operations required in the case study) and define a concrete recording strategy using HBase.

Based on the operations that the nodes in the distributed environment need to perform to record the provenance metadata, we define the methods in our

service. In chapter 4 we described these operations using the terms of the unified model PCM, but we will use PROV-DM to define the operations, in order to obtain a more general solution. As was shown in the previous section, PROV-DM is more general than our unified model and permits to express everything of it.

We use a subset of the PROV-DM model, but the terms and relations used are used respecting the model's definitions, so it is compliant. We use only a subset of the model because that is what we need to satisfy our requirements, but should there be a need to include other concepts or relations they could be easily incorporated by defining additional methods and data structures.

5.2.1 Recording Services

We will distinguish two types of services, first those in which the repository actually records provenance metadata, and second, those used to support a federated metadata repository. The latter are solely used when the provenance capture node is unable to invoke the services to record metadata in the centralized repository. In that case, it must register a minimal information so provenance can be traced by querying it.

Provenance Recording Services

We now revisit the operations performed by the provenance capture nodes and the corresponding methods using PROV-DM terminology.

1) When a data transformation is executed, the system running it should register the following information:

<transformation execution id, transformation specification id, execution context >

This can be represented by the PROV-DM wasAssociatedWith[70] relation as follows:

wasAssociatedWith(id; a, ag, pl, attrs) in PROV-N, has:

- id: an optional identifier for the association between an activity and an agent;
- activity: an identifier (a) for the activity, that represents the transformation execution id;
- agent: an optional identifier (ag) for the agent associated with the activity;
- plan: an optional identifier (pl) for the plan the agent relied on in the context of this activity, that represents the transformation specification id;
- attributes: an optional set (attrs) of attribute-value pairs representing additional information about this association of this activity with this agent, that can be used to represent information about the execution context.

2) When running workflows we face very similar problems. The workflow execution corresponds to a workflow specification, which must be registered as follows:

<workflow execution id, workflow specification id, execution context >

This is registered the PROV-DM wasAssociatedWith[70] relation analogously as the transformation executions.

3) When a workflow runs, it will follow that specified logic to invoke the different data transformations. When it calls the data transformations it is informed of their identifiers which it must register as follows:

<workflow execution id, transformation execution id, position in workflow >

This can be represented by the PROV-DM wasStartedBy[70] relation as follows:
wasStartedBy(id; a2, e, a1, t, attrs) in PROV-N, has:

- id: an optional identifier for the activity start;
- activity: an identifier (a2) for the started activity, that represents the transformation execution id;
- trigger: an optional identifier (e) for the entity triggering the activity;
- starter: an optional identifier (a1) for the activity that generated the (possibly unspecified) entity (e), that represents the workflow execution id;
- time: the optional time (t) at which the activity was started;
- attributes: an optional set (attrs) of attribute-value pairs representing additional information about this activity start, that can include information about the position of the data transformation in the workflow.

4) Using the identifiers provided the Id Creation Scheme and the Data Set Catalogue for the input and output, and the identifier assigned to the transformation execution, the Provenance Capture module will record coarse-grained provenance as follows:

<output data set id, [input data set id], transformation execution id >

This can be represented by the PROV-DM used[70] and generatedBy[70] relations as follows:
used(id; a, e, t, attrs) in PROV-N, has:

- id: an optional identifier for a usage;
- activity: an identifier (a) for the activity that used an entity, that represents the transformation execution id;
- entity: an optional identifier (e) for the entity being used, that represents the input data set id;

- time: an optional "usage time" (t), the time at which the entity started to be used;
- attributes: an optional set (attrs) of attribute-value pairs representing additional information about this usage. -

wasGeneratedBy(id; e, a, t, attrs) in PROV-N, has:

- id: an optional identifier for a generation;
- entity: an identifier (e) for a created entity, that represents the output data set id;
- activity: an optional identifier (a) for the activity that creates the entity, that represents the transformation execution id;
- time: an optional "generation time" (t), the time at which the entity was completely created;
- attributes: an optional set (attrs) of attribute-value pairs representing additional information about this generation.

5) Also, the Provenance Capture module will record fine-grained provenance as follows:

<output data item id, [input data item id], provenance semantics, transformation execution id >

This can be represented by the extended version of PROV-DM WasDerivedFrom [70] relation as follows:

wasDerivedFrom(id; e2, e1, a, g2, u1, attrs) in PROV-DM has:

- id: an optional identifier for a derivation;
- generatedEntity: the identifier (e2) of the entity generated by the derivation, that represents the output data item id;
- usedEntity: the identifier (e1) of the entity used by the derivation, that represents the input data item id;
- activity: an optional identifier (a) for the activity using and generating the above entities, that represents the transformation execution id;
- generation: an optional identifier (g2) for the generation involving the generated entity (e2) and activity (a);
- usage: an optional identifier (u1) for the usage involving the used entity (e1) and activity (a);
- attributes: an optional set (attrs) of attribute-value pairs representing additional information about this derivation, can represent the semantics of the provenance relation between input and output.

Federated Provenance Services

As was mentioned earlier, it is possible that some system cannot call the aforementioned services, in which case we need some information to be able to span provenance information across that system. To perform forward tracing over the data items used by these systems, it is necessary to know which data items were consumed by them.

For this we incorporate the possibility for the system to declare which data it consumes:

$$<system\ id,\ input\ data\ set\ id>$$

Since this is not part of our provenance model, we do not represent it using PROV-DM. For this purpose we define the following method:

systemUsed(system id, input data set id)

- system id: represents the system which is executing data transformations, the id is the one provided by the Systems Catalogue.
- input data set id: represents the data sets accessed by the system.

5.2.2 Consuming Services and Recording Strategy

Given the amount of metadata we need to manage, which was too large in traditional relational databases to be recorded [76], and is bigger if we include Hadoop data, we consider Hadoop HDFS as the best option over relational databases. Also, given the random access required when querying the metadata, we select HBase over plain HDFS, as recommended by Hadoop: The definitive guide [79]. Finally, Franke et al. [52] present a case in favour of HBase for recording provenance information.

As we mentioned earlier, one of the advantages of defining a communication interface is to be able change the internal representation of the data without affecting the provenance capture nodes. Also, given the case study specific requirements, one can design the internal representation to better cater for them. In particular, in our case study presented in chapter 5, the motivation for having provenance information is to be able to update quality attributes for already processed data when the quality attributes for their ancestral data are updated. This is a case of forward-tracing, and we will design our repository's data structure to favour this operation.

Also, we take advantage of the workings of HBase, which works at its best when how the data will be accessed is taken into account when designing the schemas. This is actually mentioned as the most important consideration since "all access is via primary key so the key design should lend itself to how the data is going to be queried" [79].

HBase is a distributed column-oriented database built on top of HDFS. designed to provide real-time read/write random-access to very large data sets. In HBase, data is stored in tables, which have rows and columns. A row in HBase consists of a row key and one or more columns with values associated with them. Rows are sorted alphabetically by the row key as they are stored. A column in HBase consists of a column family and a column qualifier, which are delimited by a : (colon) character. Column families physically co-locate a set

Table: WasAssociatedWith		
Row Key:	activity	
Family:	agent:	columns: agent, id
	plan:	columns: [ ] → plan
	attributes:	columns: attribute → value

Table 5.1: HBase Table WasAssociatedWith.

of columns and their values, often for performance reasons. Finally, a cell is a combination of row, column family, and column qualifier, and contains a value and a timestamp, which represents the value's version [59].

Taking all the requirements into account, we propose to have the following 5 tables, one for each of the defined operations:

- WasAssociatedWith,
- WasStartedBy,
- Used,
- WasGeneratedBy,
- WasDerivedFrom.

The table WasAssociatedWith (table 5.1) holds the information sent via the method wasAssociatedWith(id; activity, agent, plan, attributes) when a data transformation or a workflow is executed.

Since we want to access it for forward-tracing operations, we want to favour queries over the transformation or workflow execution, i.e. the PROV-DM Activity, we select the activity as the row key. There will be three column families: agent, plan and attributes. In the agent family, the column keys are agent and id, where it is possible to record respectively the identifier for the agent and the identifier for the association between the activity and the agent. In the plan family, we use only one column with the empty qualifier to store the identifier for the plan. In the attributes family, a column is created for each attribute-value pair, using the attribute as column key.

The table WasStartedBy (table 5.2) holds the information sent via the method wasStartedBy(id; started activity, entity (trigger), triggering activity, time, attributes) when a workflow runs and it invokes the different data transformations. The row key is the started activity in order to favour forward-tracing. In the column activity:triggering activity records the activity that triggered the started activity. In the time family, we use only one column with the empty qualifier to store the time at which the activity was started. The attributes column family works analogously as in the previous table.

The table Used (table 5.3) holds the information sent via the method used(id; activity, used entity, time, attributes) when a data transformation consumes data. The key is composed of the identifier for the used entity and that of the activity that used it. This key takes advantage of the partial key scan functionality offered by HBase, which permits to obtain all the rows between a start and end key, thus all the entity-activity pairs for a given entity identifier. It is therefore easy to obtain all the activities that used a certain entity, thus favouring forward-tracing operations.

Table: WasStartedBy		
Row Key:	started activity	
Family:	activity:	columns: triggering activity, id
	time:	columns: [ ] → time
	attributes:	columns: attribute → value

Table 5.2: HBase Table WasStartedBy.

Table: Used		
Row Key:	entity - activity	
Family:	id:	columns: [ ] → id
	time:	columns: [ ] → time
	attributes:	columns: attribute → value

Table 5.3: HBase Table Used.

The table WasGeneratedBy (table 5.4) holds the information sent via the method wasGeneratedBy(id; generated entity, activity, time, attributes) when a data transformation generates data. The key is composed of the identifier for the activity that generated the entity and the identifier for the generated entity. This key also takes advantage of the partial key scan functionality, facilitating the query of all the entities generated by a certain activity, thus favouring forward-tracing operations.

The table WasDerivedFrom (table 5.5) holds the information sent via the method wasDerivedFrom(id; generated entity, used entity, activity, generation id, usage id, attributes) when data items are derived from other data items. The row key is composed of the identifier for the used entity and the identifier for the generated entity, permitting via partial key scan to query the entities derived from a given one.

As for maintaining the federated provenance repository, we showed that it is only necessary that the system that is consuming data informs it to the central repository. For that, it calls the method systemConsumed(system id, input data set id). This can be recorded in the following HBase table SystemUsed (table 5.6). The row key is composed by the identifier for the entity used and the identifier for the system which used it. This favours forward-tracing since it is easy to query for a given entity if it was used and by whom.

Table: WasGeneratedBy		
Row Key:	activity - entity	
Family:	id:	columns: [ ] → id
	time:	columns: [ ] → time
	attributes:	columns: attribute → value

Table 5.4: HBase Table WasGeneratedBy.

Table: WasDerivedFrom		
Row Key:	used entity - generated entity	
Family:	activity:	columns: activity, generation id, usage id
	attributes:	columns: attribute → value

Table 5.5: HBase Table WasDerivedFrom.

Table: SystemUsed		
Row Key:	used entity - system	
Family:	usage:	[ ] → timestamp

Table 5.6: HBase Table SystemUsed.

5.3 Implementing Provenance

In this section we present the aspects of the implementation of provenance, in particular the type of provenance that can be obtained in different contexts, and how the provenance information only improves with more restrictions to the system. Also, we show that the notions of how- and where-provenance are dependent on the existence of mappings to define the data transformations. Then we review the surveyed works to see the details of their implementations and close the section with a summary of the reviewed solutions.

5.3.1 Automatic Provenance Capture

As we mentioned earlier, a common objective of the reviewed works is that provenance capture and recording is done in a completely transparent manner and without imposing any restrictions either to the user or the developer.

This objective, though seldom stated in the literature, is always present and solutions where the user is required to manually record provenance information are rare (generally the older works). Requiring the user to do this is not only error-prone but also difficult and expensive, and therefore the reviewed solutions which follow this approach offer only coarse-grained provenance.

Also, the transparency from the point of view of the developer is sought and it is rare for developers to be requested to explicitly invoke APIs to record provenance. Having the developer of data transformations explicitly addressing provenance has several inconveniences: he is distracted from his main objective of developing the data transformation; he could do it unsystematically; it becomes another thing to test; etc. Ideally, the provenance capture solution should occur outside of the transformation, so that it is done consistently. Also, having the provenance capture defined independently of the transformations would permit to redefine its criteria without affecting the transformations.

There are some aspects of provenance that cannot be expected to be captured in an automatic and transparent way, such as the name, description or objective of a data transformation. On the other hand, the capture of the causal relation between data items or data collections, which are input and output to data transformations, is expected to be captured as automatically and transparently as possible.

Regarding the transparency of provenance capture, we identify the following situations in the reviewed works:

I. No restrictions imposed on the type of transformations, the user is requested to declare the causal relations, it results in coarse-grained provenance [51, 72].

II. No restrictions imposed on the type of transformations, the developer is requested to invoke an API declaring the causal relations. We reviewed one solution following this approach which offers coarse-grained provenance [53]. However, if addressed by the developer fine-grained provenance could be offered.

III. No restrictions imposed on the type of transformations, a wrapper is applied over the data transformations, it results in coarse-grained provenance [46, 44].

IV. Imposing certain restrictions on the type of transformations allowed, a wrapper is applied over the data transformations, it results in fine-grained provenance, in particular, why-provenance. The restrictions depend on the specific context, but could be: belonging to a certain transformation class or the existence of schema mappings [46]; that it be a Constant Mapping Operation [60], or in a MapReduce context requiring that both map and reduce functions be deterministic and pure [61, 73].

V. Restricting to declaratively defined transformations, it results in fine-grained provenance, in particular, why-, where- and how-provenance [37, 65, 45, 35, 55, 33].

We summarize this in the table 5.7. Situations I and II are the most atypical, which require the user to register provenance or the developer to consider it when developing the transformations. The remaining situations (III-V) are the most common ones as well as the most transparent. In these we can observe how the results improve (finer granularity or better notions of provenance) only when the imposed restrictions increase.

Data Transformations' Specifications and Notions of Provenance

When explaining the causal relation between inputs and outputs of a data transformation, the most complete information is provided by the notions of where- and how-provenance, which explain not only which (why-provenance) data items were used as input but how they were used.

It must be noted, however, that these notions of provenance are strongly dependent on the use of mappings (e.g. queries) to declaratively define the transformations. A transformation is "any procedure that takes data sets as input and produces data sets as output" [46, 61]. As such, it may have or exhibit schema mappings, or be completely defined by mappings [46]. A schema mapping is a "specification that describes the relationships between schemas at a high level" [62]. Rahm and Bernstein [74] present a more general definition stating that a mapping is "a set of mapping elements, each of which indicates that certain elements of schema S1 are mapped to certain elements in S2, where

Situation	Developer	User	Restrictions	Result
I	None	Register provenance	None	Coarse Granularity
II	Invoke API	None	None	Fine or Coarse Granularity
III	None	None	None	Coarse Granularity
IV	None	None	Restricted types of transformations	Fine Granularity (only why-p)
V	None	None	Declaratively defined	Fine Granularity (why-, where- and how-p)

Table 5.7: Characteristics of Provenance.

each mapping element can have a mapping expression which specifies how the S1 and S2 elements are related". They also define a schema in a very general way, as "a set of elements connected by some structure".

These better notions of provenance are obtained because of the existence of the mapping, from which both the source of a tuple's value (where-p) or its composition (how-p) are obtained. If the mapping was not available, it would be necessary to do a reverse engineering process to obtain this kind of provenance. On the other hand, while why-p can be derived from the mapping, its existence is not necessary and solutions do exist where why-p is computed for non-declarative transformations.

We point out that if provenance was explicitly addressed by someone implementing a transformation (e.g. writing code), specific code could be included to permit the capture and record of metadata for where- and how-provenance, thus solving provenance inside the transformation. However, as seen above, this approach is not common in the reviewed literature.

Here we highlight the important role played by the query, which is a mapping between schemas, i.e. a high-level declarative specification of the transformation taking place. Even though the transformation is considered as given and provenance is solved outside of it, there is this vital piece of metadata which permits to obtain a much more specific provenance than just why-provenance, namely where- and how-provenance.

5.3.2 Reviewed works

In this section we review the surveyed works to see the details of their implementations, in particular if they are based on metadata or a tracing procedure, how automatic the solutions are and the restrictions they impose to the users, and how they solve the data identification problem. The works are ordered by

publication year and alphabetically by first author.

Buneman et al. (2001)

Buneman et al. [37] propose a framework to trace backward provenance based on a syntactic analysis of the data transformations in the context of databases and transformations defined by SPJU (Select-Join-Projection-Union) queries.

The representation of where-provenance requires the identification of locations in the source data, so data items need a way be identified. The authors assume that data items can be uniquely identified, and if that is not the case, they state that collections are treated as sets, identifying the data items by their value. As a basis for describing where-provenance the authors use an edge-labeled tree data model where "the location of any piece of data can be uniquely described and determined by a path". Relations are cast to the model by using the keys as edge labels. If there is no key for the relation, the tuples are modelled as a set, i.e. the entire tuple becomes an edge label. The authors state that object-oriented or semi structured databases with object identifiers for all structures can be expressed using the mentioned model.

As mentioned earlier, the authors' approach is based on the syntactic analysis of queries, so it is limited to data transformations declaratively defined. Although not addressed by the authors, this approach could be used to compute provenance both previously and on demand, i.e. a metadata approach or a tracing procedure approach. Yet, a possible provenance recording mechanism is not presented in the article.

Since everything needed to compute provenance is available in the query, nothing is needed from the user for the system to work.

Although not stated by the authors, it is assumed that the data sources remain unchanged. If this was not the case, the algorithms proposed would not work.

Frew et al. (2001)

Frew et al. [53] have developed a metadata recording framework called Earth System Science Workbench (ESSW). The ESSW system has a client-server architecture, where a researcher's workstation acts as a client sending the metadata and causal associations to the server which in turn records it. This permits to record provenance for transformations executing in different machines, where data items may be shared or not.

The authors claim that provenance capture is transparent or nonintrusive "because it relies on scripting", where "wrappers or ancillary scripts log science object metadata, with minimal alterations to existing processing methods". However, provenance capture is not really transparent because "the scripts that carry out each experiment either invoke commands that collect the values for science object metadata or explicitly provide these values".

The system works in a context where input and output data may be image files. In this context, the authors identify two problems: "first, there is no intrinsic connection between a file's name and its contents- the metadata encoded in a file's path name is always vulnerable to a file's being moved, renamed, or re- written. Second, there is no simple, portable way to maintain multiple path names to the same file." These problems are those of identifying the data item

(the file) and that of maintaining the id-location association. These problems are addressed by them by using a MD5 hash digest as the identifier for the files.

The authors claim ESSW may be used to characterize workflows at whatever level of granularity needed, but even though it would be possible to refer to fine-grain data items, it would not be possible to declare the relations between these data items, because all that can be declared is that they were either input or output to a certain experiment, so only coarse-grained provenance is actually recorded.

Marathe (2001)

Marathe [65] presents a "lineage tracing algorithm" called SUB-pushdown for data transformations over array data which offers fine-grained provenance, in particular why-provenance.

SUB-pushdown is restricted to array manipulations expressed using AML (a declarative language). The author points out that "an interesting research question is how to trace data lineage in arbitrary array computations".

This algorithm uses the definition of the array transformations as input, therefore it needs it to be available for use. It also requires the input arrays to be available, for it to be able to identify the participating data items in them. Data items are identified by their positions in the arrays.

Foster et al. (2002)

Foster et al. [51] have developed a prototype for a system called Chimera, which combines a virtual data catalogue, for representing data derivation procedures and derived data, with a virtual data language interpreter that translates user requests into data definition and query operations on the database.

The architecture of the Chimera virtual data system comprises two principal components: a virtual data catalogue (VDC; this implements the Chimera virtual data schema) and the virtual data language interpreter, which translates the calls to the virtual data catalogue operations into SQL queries to the database (inserts, updates, queries).

The paper does not address the question of how the data maintained within the Chimera system is produced, but state that information about transformations and derivations could be manually recorded by the user and/or created by monitoring job execution, among others.

Regarding the identification and location of transformations and data files, the authors in both cases create unique identifiers for them and separate the identification and location problems by creating a mapping component. In the case of logical transformations, they are characterized by an identifying name, the namespace within which the name is unique, and a version number. Then, the logical transformation can be associated to many physical transformations. In the case of data files, they are considered as logical files, named by a logical file name (LFN); then a "separate replica catalogue" or "replica location service" would be used to map from logical file names to physical file location(s).

Cui et al. (2000, 2003)

Cui et al. [45, 47, 46], working in the context of data warehousing, present an algorithm to trace fine-grained provenance for both relational views and for

non-declarative transformations.

The authors define schema mapping in the context of a transformation T with input schema A and output schema B. The mapping is defined as a function from tuples of attribute values in one schema to tuples of attribute values in the other schema; where its domain can be either a subset of attributes of A or a subset of attributes of B, and the range a subset of attributes of B or a subset of attributes of A, respectively; and where some other conditions are met.

It is possible that the complete transformation is specified via mappings (the transformation is specified as a standard relational operator or view). On the other hand, some transformations are simply procedural code and lack the mapping specification. However, Cui et al. [46] say that transformations "often lie between these two extremes, they are not standard relational operators, but they have some known structure or properties that can help" to identify and trace data provenance.

Given a transformation T, an input set I and a subset of the output set which we call O*, the authors propose a series of algorithms to determine the provenance of O*, depending on the class of the transformation T. The algorithm can be more or less efficient depending on the presence of schema mappings in the transformation and on the properties these mappings may have.

For most of the algorithms, it is assumed that the input to the already executed transformation is readily available. If no schema mapping is present, it is necessary to rerun the transformation to obtain the provenance of O*; if one mapping is present, it can help reduce the set of possible provenance elements from the input set; if more mappings are present, these can reduce even more the set of possible provenance elements. If the mappings enjoy certain properties, it may suffice to only recall the mappings instead of the whole transformation. For the particular case when the transformation is entirely defined by mappings, i.e. relational views, a special tracing procedure is given based on the view definition. Regarding this, the authors point out that the data "sources may be inaccessible, expensive to access, expensive to transfer data from, and/or inconsistent with the views at the warehouse", so the problem could be avoided by storing auxiliary views in the warehouse.

On the other hand, for one particular case of transformation it is not necessary to have available the input set. When a tracing procedure for the transformation (this is not necessarily the inverse of the transformation) is provided, it may, or not, be necessary to have available the input set.

When the transformation cannot be classified in any of the categories defined (i.e. it is neither a dispatcher nor an aggregator) and does not have a provided provenance tracing procedure, the authors call it a black-box transformation. In this case, the entire input data set is the provenance of each output item, thus resulting in coarse-grained provenance.

The problem of identifying data items is not addressed by the authors. The presented tracing procedures return the actual data, so we can say that data is identified by its value inside a certain set.

Pancerella et al. (2003)

Pancerella et al. [72] work in the context of chemical sciences and have developed a system called Collaboratory for the Multi-scale Chemical Sciences (CMCS) which offers a suite of tools for managing data and metadata and visualizing

provenance relationships between data entries with coarse granularity. The system is basically a repository of metadata that is used to describe files (that is, for data sets) and their relations to projects and to other data sets. The authors state that this metadata could be loaded in four ways: using the Scientific Annotation Middleware (SAM) which runs a user defined XSLT to translate metadata from another schema, manually via the web application, using the DAV protocol or an API.

However, the problem of capturing the provenance metadata is not addressed by the authors. All that is said is that the API "allows scientists to easily write programs to add/edit metadata and to integrate with existing and new chemical science applications."

Regarding the identification of data collections or resources within the CMCS data repository, URIs are used to reference them.

Bhagwat et al. (2005)

Bhagwat et al. [35] work in the scientific domain and have developed an annotation management system for relational databases in which every piece of data in a relation is assumed to have zero or more annotations associated with it and annotations are propagated along, from the source to the output, as data is being moved through a query. When a query (which reads and saves data) is executed the data item's annotations are propagated using the propagation scheme selected by the user.

The system offers three annotations propagation schemes: the default, the default-all and a custom one. The default scheme uses where-provenance as the basis for propagating annotations. If an output piece of data d' is copied from an input piece of data d, then the annotations associated with d are propagated to d'. Because that the way annotations are propagated is dependent on the way a query is written (two equivalent queries may propagate annotations differently), an alternative method of propagating annotations is offered, called the default-all scheme, which propagates annotations according to where data is copied from in all equivalent formulations of the given query. Also, a custom mechanism is offered, where the user defines which annotations are copied to which location.

Regarding why-provenance, the authors set as future work to extend their system to offer a propagation scheme based on why-provenance. They state that this scheme should return the set of all annotations in an output location if it occurs in the same output location in the results of all equivalent queries. Still, they state that it needs to be investigated whether a query basis can be generated for such propagation scheme. However, a simpler approach could be taken, not considering all the equivalent queries but just the actual query, which is the approach taken by Cui et al. [45]. Also, in this work, for the notion of where-provenance, two approaches are considered, one for the actual query and one for all the equivalent ones.

In this system, the metadata necessary to "answer provenance queries" (the way understood by them) is computed when the transformation takes place and is ready for use whenever needed. In this scheme, answering provenance is "fast" since no traceability function needs to be executed. However, the transformations (query execution time) are penalized by the propagation of the annotations (to a large degree in the default-all scheme according to the experiments conducted by Bhagwat et al.). Also, even though nothing is said

about it in the article, the size of the metadata it can be potentially large with respect to the amount of data. If the default scheme is used for a 100% annotated database (i.e. every value has one annotation), with every query, the size of the metadata grows proportionally with the size of the data. If the default-all scheme is used for a 100% annotated database, with every query, the size of metadata will grow proportionally more than the size of the data.

No concept of data identification exists in this work. Every time the data is copied to a new table inside the same database, all its provenance information is replicated, and it is not clear each version of the "same" data is considered to be different.

Green et al. (2007)

Green et al. [55] show the limitation of why-provenance and develop the notion of how-provenance to overcome this, where a polynomial is used to show how the input data items were used to construct an output data item.

The notion of how-provenance is strongly related to the existence of mappings to define the transformations, since it is from the mappings that how-provenance information is derived. In particular, the authors work in the context of relational databases, where data transformations are restricted to relational algebra. They propose to derive from the relational query, for every output tuple, a polynomial that represents not only which tuples were used to create the output tuple, but also how they were used.

The polynomial is expressed using the "ids" of the input. It is thus assumed by the authors that there exist identifiers for the data items. However, they do not state how they should be assigned.

The authors do not mention if any prototype of their solution has been developed. Should a system like this be developed, it could be implemented either with a metadata approach or a tracing procedure approach. In either case the how-provenance metadata could arguably be transparently captured as long as only relational algebra is used to define the transformations. Yet, no estimates are given of the space needed to record neither the polynomial nor the time needed to compute it.

Amsterdamer et al. (2011)

Amsterdamer et al. [33] work in the context of workflows whose transformations (referred to as modules in this work) are Pig Latin queries. In this context they propose a solution (and have built a prototype) that permits to compute fine-grained data provenance (both for backward- and forward-tracing) using metadata. The provenance derivation scheme they propose is based on the fact that Pig Latin query constructs can be translated into (the bag semantics version of) nested relational calculus, hence to a declarative specification, i.e. a mapping. Their solution provides how-provenance for transformations specified using Pig Latin, however, when the transformation is not specified this way, it provides only why-provenance.

Their system architecture is based on two sub-systems: a Provenance Tracker and Query Processor. The Provenance Tracker is responsible for tracking provenance for tuples that are generated over the course of workflow execution, and

to write the output to the file system. This output is read by the Query Processor sub-system, which builds the provenance graph in memory, is responsible of answering provenance queries.

The authors say that the Provenance Tracker does not involve any modifications to the Pig Latin engine, that it is implemented using Pig Latin statements invoked during workflow execution. However, no further details are given of how provenance is actually captured, nor how the workflow execution is registered. Also, the issue of how why-provenance for non-Pig Latin transformations is captured, is not addressed.

Other unmentioned aspects are: how are records identified (this is necessary to express the semirings as shown by Green et al. [55]) and how is metadata stored in the file system.

Regarding the restriction of acyclicity in the workflows, the authors explain that it "dealing with recursive workflows would introduce potential nontermination in the semantics and, to the best of our knowledge, this is still an unexplored area from the perspective of provenance."

Crawl et al. (2011)

Crawl et al. [78, 32, 44] work in the context of scientific workflows, where they have developed a workflow management systems called Kepler, which they have integrated with Hadoop, creating Kepler+Hadoop. The Kepler system permits to model and execute workflows and the Kepler+Hadoop version permits the transformations (referred to as actors in Kepler) to be MapReduce jobs. They provide a metadata based framework to compute coarse-grained provenance.

In this system, workflows are defined and executed from Kepler (from outside of Hadoop). Data is copied from Kepler to the HDFS to be processed by the MapReduce jobs and when finished it is copied back. Provenance metadata is stored outside of Hadoop in a MySQL Cluster.

Kepler adopts an actor-oriented modeling paradigm, where the workflow is composed of actors, which can consume or produce data, called tokens. Workflows and its actors must be defined and executed through the Kepler system, which can trigger the metadata collection module (called Provenance Recorder) if the user so wants (it is an optional feature). To give provenance support for MapReduce actors, the Kepler+Hadoop system presents extensions to the original system, following a wrapper approach to capture provenance automatically, with coarse-granularity.

When a map or reduce task executes, its input and output data (to which they refer to as dependencies) are identified and recorded. These dependencies between the data items and the executions is registered in a MySQL cluster. Each data item is assigned a unique identifier (which is used to declare the dependencies) and the data item with its identifier is also recorded in this database.

The identifier for a data item is constructed in part by the provenance repository and in part by the executing node. The identifier is composed of three identifiers, (R, S, N): First, the identifier of the workflow execution -which they call R-, provided by the provenance repository, then S is the identifier of the MapReduce task within the MapReduce job and finally N is the artifact number within S. The latter two are provided by the executing node. An identifier in this format is unique since R is different for each workflow execution, S is generated by Hadoop to uniquely identify each MapReduce task, and N is unique

within the MapReduce task S.

Huq et al. (2011)

Huq et al. [60] work in the context of stream data processing for data coming from sensors. They propose a hybrid approach (metadata and tracing procedure) which permits to compute fine-grained data provenance. Their approach is based on a temporal data model, which, by adding a temporal attribute (e.g. timestamp) to each data item, allows to retrieve the overall database state at any point in time. This, together with what they call "coarse-grained provenance of the transformation" (information regarding the transformation: its classification, its sources and characteristics regarding the window processing and execution trigger), allows them to construct provenance. Therefore, some metadata is recorded for every tuple, but this is not provenance metadata explicitly and a tracing procedure is used to compute provenance. Hence, we classify it as a hybrid approach.

The computation is done as follows: first they calculate (or filter) the set of the input tuples which form the processing window for the tuple for which provenance is computed. Then, based on the temporal ordering of the tuples and the transformation metadata, the contributing tuples can be identified.

This tracing procedure can be applied to SQL operations or "generic functions" (e.g. interpolate, extrapolate), with the requirement that it be a Constant Mapping Operation. This means that it has a fixed ratio of mapping from input to output tuples per window, i.e. $1:1$, $n:1$, $n:m$. For variable mapping operations (i.e. those that have not any fixed mapping ratio from input to output tuples), the solution cannot be applied directly. They mention as a possible solution could be to transform these operations into constant mapping operations by introducing NULL tuples in the output, however this idea is not developed in their work nor is an estimation of the storage overhead given.

They analyse the storage consumption of their solution for their case study (i.e. the interpolation of sensor data, which is a constant mapping operation), comparing the storage overhead against the size of the actual sensor data and against an explicit provenance metadata solution. For that particular example, their hybrid approach takes at least 4 times less storage space than the explicit approach. Also, the storage space needed by the metadata of their approach is less than half of the space required for the actual data.

No analysis is presented regarding the time it takes to compute provenance using their solution.

Ikeda et al. (2011)

Ikeda et al. [61, 73] have built a prototype system called RAMP as an extension to Hadoop, to support the automatic capture and recording of fine-grained metadata-based provenance (both for backward and forward tracing). Their approach consists of transparently wrapping Hadoop MapReduce functions, to capture and record provenance metadata. The system assumes that every tuple has a globally unique identifier, so when a transformation is executed, the identifiers of the participating tuples are copied to the created tuples. For map functions -which produce zero or more output elements independently for each element in its input set-, they extract the unique identifier of each input element

that produces one or more output elements, and add that ID to each of the output elements. For reduce functions -which take an input data set I in which each element is a key-value pair, and return zero or more output elements independently for each group of elements in I with the same key-, they keep track of the grouping key for each input group, and add that key to each output element produced by the group.

The solution requires that every tuple has a globally unique identifier. RAMP can receive a scheme to assign them, or it can apply its default one. The default scheme, which works when data sets are stored in files, consists of using (filename, offset) as a default unique ID for each data element.

RAMP stores provenance metadata separately from the input and output data. It is stored in files, in particular using one provenance data file for output data file. The provenance data file associates output item identifiers with input item identifiers. The latter are composed of an input file number and and offset, so RAMP also maintains a dictionary with the actual filenames.

RAMP implements the capture of provenance through a wrapping scheme. In particular, by wrapping the following components of Hadoop: Record-reader, Mapper, Combiner, Reducer and Record-writer. The Record-reader's wrapper assigns a unique ID p to each input element, the Mapper is wrapped so it is unaware of this but its output records contain the ID. The Reducer's wrapper iterates over all annotated map output elements with the same key k and feeds record to the reducer. While doing this, the Reducer's wrapper stores the map provenance, i.e. the relation between the input record and the mapper output, namely (k, p). Then the Record-writer's wrapper assigns a unique identifier q to each produced record and stores the reduce provenance, i.e. the relation between the output record and the grouping key of the reduce function, namely (q, k).

The granularity of transformations for which provenance is captured is a MapReduce job. In particular, no intermediate data is stored between the Map and Reduce functions. Also, a Map function which is not followed by a reduce function, and a Reduce function which is not preceded by a Map function, are treated as MapReduce jobs.

RAMP requires that both map and reduce functions are deterministic and pure. A map function is considered pure if it produces zero or more output elements independently for each element in its input set. A reduce function is considered pure if takes an input data set I in which each element is a key-value pair, and returns zero or more output elements independently for each group of elements in I with the same key. The key aspect of these definitions is the word independently, which means these functions do not buffer the input or otherwise use "side-effect" temporary storage, which would make it impossible to associate input and output data items correctly.

The authors conduct experiments to measure the performance of their application regarding two aspects: first, the time and space overhead of provenance capture, and second the time to compute why-provenance (backward-trace).

Depending on the type of the transformation being traced (in particular, due to its multiplicity), time and space overhead during provenance capture can vary considerably. For the Wordcount problem (a many-one transformation, where each intermediate and output data element is annotated with many input element IDs), provenance capture incurred a 72-76% time overhead and the number of annotations per data element increases with input size. For the

Terasort problem (a one-one transformation, where each intermediate and output data element is annotated with exactly one input element ID), provenance capture incurred a 16-20% time overhead and a 19-21% space overhead.

The time needed to compute provenance also varied greatly from one problem to another. The authors report that for Wordcount, tracing one element took approximately 1, 3, and 5 minutes, for 100, 300, and 500 GB input data sizes respectively. For Terasort, tracing one element took approximately 1.5 seconds for all input data sizes.

RAMP's provenance recording scheme is biased towards backward tracing, which the authors assume is a more frequent operation than forward tracing. They state that "without auxiliary structures, each forward-tracing step would require a complete scan of the map provenance, which is not sorted on input element IDs".

Akoush et al. (2013)

Akoush et al. [31] present a modified version of Hadoop -called HadoopProv- which offers capturing and recording of fine-grained why-provenance in MapReduce jobs. HadoopProv aims to minimize provenance capture overheads while the MapReduce jobs execute and hence follows a hybrid approach to provenance. They do this by recording causal relationship between input and output records in Map and Reduce phases separately. This is not the causal relationship between input and output records of the MapReduce job, but the latter is computed with this information on demand, thus we consider it a hybrid approach. This provenance information is stored in separate files.

More specifically, when a Map task executes, it reads records from an input file split and applies a user-defined Map function to emit intermediate key-value pairs. Here, HadoopProv captures and records the association between each emitted intermediate key and the locations of corresponding input records that participated in its creation. When a Reduce task executes over the intermediate key-value pairs, HadoopProv captures and records, for each output record, the association between its location at the output file and the intermediate key given to that task. Then, at query time, the entire provenance graph can be constructed by joining the Map and Reduce provenance files on all matching intermediate keys.

Provenance capture is addressed by the system itself, since this version of Hadoop has been modified in order for it to capture causal relationships between input and output records. However, no details are given of how it is done and it is not mentioned by the authors if there is any restriction on the kind of MapReduce transformations for which the system works. It would appear that it would have the same restrictions as the solution proposed by Ikeda et al. [61].

Regarding the identification of the data items, the system uses their location in the data sets as their identifier. This has the drawback that the provenance metadata would be "broken" should the data set be reordered.

5.3.3 Summary of reviewed works

In table 5.8 we summarize the survey of existing solutions in the literature, putting special attention in the following characteristics:

- First, if the solution is based on precomputing metadata, on applying a tracing procedure or on a hybrid approach.
- Second, if the system requires the intervention of the user or it can automatically obtain the input to compute provenance.
- Another characteristic refers to restrictions imposed on the data transformations for the system to work, e.g. if the provenance system supports only transformations which are defined declaratively by a high level language (i.e. using mappings, e.g. a SQL query), or it only works for a certain type of transformations, etc.
- The fourth and fifth dimensions refer to what the system offers: the granularity of provenance offered (Fine or Coarse) and the notion of provenance offered (i.e. why-, where- or how-provenance).
- The sixth characteristic refers to the identifying scheme used.

The system developed by Foster et al. [51] offers among other things a metadata framework, but do not address the question of how the metadata is maintained. They just say that it could be manually recorded by the user and/or created by monitoring job execution, so we classify it as manual.

The work of Cui et al. has three entries in the table because they offer different solutions if the transformation is defined declaratively or not or if the transformation has certain properties. In the first case, the schema mapping (i.e. the query) is enough to compute provenance, in the second one the user is required to specify the properties that are present in the transformations. If no properties are present, the solution only offers coarse-grained provenance.

We can observe from the reviewed literature that the way provenance is computed (via metadata or a tracing procedure, i.e. pre- or post-computed respectively) and the way the transformations are specified (declaratively or not) are two independent dimensions.

We have reviewed works where metadata is pre-computed for both declaratively-defined and non-declarative transformations, e.g. Bhagwat et al. [35] and Ikeda et al. [61] respectively. On the other hand, we have reviewed works which follow a post-computation approach for both declaratively-defined and non-declarative transformations, e.g. Cui et al. [45] and Cui et al. [46] respectively.

5.4 Provenance Capture in Hadoop

In this section we show how the solution developed by Ikeda et al. [61] can be adapted to work with the presented framework, so it can work as one of the nodes capturing and sending provenance information to the central repository in our distributed environment. This solution offers fine-grained provenance for MapReduce jobs following a metadata approach.

To adapt this solution to work within the framework we only need to:

- Change where it records the provenance metadata, instead of recording it in a local file for it to send the metadata to the repository.
- The data identifying scheme should follow the framework's definition. We point out the original identifying scheme of Ikeda et al. consists of using

Reviewed Works	MD / Tr. proc.	Prov. Capture	Data Transfor-mations	Gran.	Notion of prov.	Id. scheme
Buneman et al. [37]	Tracing proc.	Auto	Decl. Defined (query)	F	Why-p, Where-p (limited)	Location as id.
Frew et al. [53]	MD	Invoke API	No restrict.	C	-	MD5 hash digest
Marathe [65]	Tracing proc.	Auto	Decl. Defined (query)	F	Why-p	Location as id.
Foster et al. [51]	MD	Manual	No restrict.	C	-	Creates ids
Cui et al. (Rel. Op.)[45]	Tracing proc.	Auto	Decl. Defined (query)	F	Why-p	Value as id.
Cui et al. (Gen. Tr.) [46]	Tracing proc.	Auto	Known class of data transf.	F	Why-p	Value as id.
Cui et al. (Gen. Tr.) [46]	Tracing proc.	Auto	No restrict.	C	-	-
Pancerella et al. [72]	MD	Manual	No restrict.	C	-	URIs
Bhagwat et al. [35]	MD	Auto	Decl. Defined (query)	F	Where-p (similar)	Not mentioned
Green et al. [55]	Not specified (could be either)	Auto	Decl. Defined (query)	F	How-p, Why-p	Not mentioned
Amsterdamer et al. [33]	MD	Auto	Decl. Defined (Pig Latin)	F	How-p, Why-p	Not mentioned
Crawl et al. [44]	MD	Auto	No restrict.	C	-	Creates ids
Huq et al. [60]	Hybrid	Auto	Constant mapping operations	F	Why-p	Not mentioned
Ikeda et al. [61]	MD	Auto	Pure Map R. functions	F	Why-p	Location as id.
Akoush et al. [31]	Hybrid	Auto	Pure Map R. functions	F	Why-p	Location as id.

Table 5.8: Summary of reviewed works.

the file name and the offset as identifier for the data items, which can easily be adapted to comply with the framework by assigning an identifier to each file which is unique in the system.

Performing these changes, we have the same solution but now extended to compute provenance in a distributed environment. We point out that the solution still has the same limitations regarding locations used as identifiers. This could be raised by incorporating the functions of the Id Creation Scheme, which would maintain the id-location relation.

This solution cannot address the identification of data transformations. Therefore, when sending the provenance metadata to the repository, this information will not be sent. To overcome this, it is necessary to maintain a Data Transformation Catalogue, where information about the MapReduce jobs is defined.

5.5 Provenance Capture in Relational Databases

Similarly, to compute provenance in a relational database, we show how to adapt one of the reviewed solutions to work within our framework. We select the solution developed by Cui et al. [47] which offers fine-grained provenance for data transformations which are relational queries.

The solution developed by Cui et al. is based on a tracing procedure which is executed on demand to offer backward-tracing provenance. Should we want to compute only backward-tracing provenance, this could be reused as it is, exploiting the possibility that the repository can work in a federated fashion. What needs to be addressed though, is data identification. It is necessary that the provenance capture function can maintain the identifiers for data items, whether the data are created inside the relational database or outside of it -e.g.: copied from Hadoop-. For example, if data produced inside of Hadoop was copied to the relational database to be used as input for data transformations, the provenance capture function should maintain the identifiers for those data items, and notify the provenance repository that it is using them. Therefore, when performing forward-tracing provenance for a data item inside Hadoop, if one of its descendant data products was copied to the relational database and registered as used by it, we know we have to ask the local provenance solution for that part of the provenance graph.

However, in our case study we want to perform forward-tracing, so the solution needs to be adapted to record provenance metadata in the centralized repository. This could be done by performing the tracing procedure over each created data item and recording the returned information.

This solution does not address data transformations either, so it will not record provenance metadata about them. To overcome this, it is necessary to maintain a Data Transformation Catalogue, where information about the relational queries is defined.

5.6 Conclusions

Our framework is based on a centralized metadata repository, where provenance capture occurs in different nodes in the distributed environment and is sent to the metadata repository. We have presented a solution for the Provenance

repository and have shown how to capture provenance in Hadoop and relational databases.

First, we reviewed the existing provenance models for provenance interoperability, which showed the most widely used models are OPM and PROV-DM. We analysed how fit both models are to represent PCM, the provenance model built in section 4.1, and found the OPM model has limitations that do not permit to represent every aspect of PCM. On the other hand, using PROV-DM and its mechanisms to define extended structures, it is possible to represent everything of our model, so we select it for use.

Then we presented a concrete implementation for our metadata repository (based on the operations required in the case study), defined the services using a subset of PROV-DM terms and defined a concrete recording strategy using HBase, in order to manage large volumes of provenance metadata.

We have also presented general aspects of the implementation of provenance, and also we have shown how to modify the solutions developed by Ikeda et al. [61] and Cui et al. [47] so that they can work within the framework and thus result in improved versions, incorporating the functionality of offering provenance outside of the original scenarios.

Using these modified solutions within the proposed framework it is possible to compute provenance in a distributed scenario of Hadoop and relational databases.

Chapter 6

Two Application Scenarios

In this chapter we present other scenarios and show how the proposed framework and the metadata repository can be adapted to satisfy the different requirements in the different use scenarios.

6.1 Tracing products of industrial processes

This scenario is set in an industrial process, where raw materials are used to produce some resulting products, and it is necessary to keep track of which input and output batches.

The industrial process we consider is that of producing biodiesel from vegetable oil, but it could be any process. In the processing plant, tankers are received with the oil to be used as input, where each tanker arrival has its identifier. In this scenario exist basically two industrial processes: first, process A, which is fed with the oil in the incoming tankers and produces an intermediate product which is stored in tanks. Secondly, process B is fed with the aforementioned intermediate product, which results in the final product of biodiesel, also stored in tanks.

There are two tracing requirements. First, when a problem is detected in process B, it is required to know which oil tankers were input to the running process. Second, it is common to receive information about the oil tankers long after they were used, so it is necessary to identify which biodiesel tanks were produced based on them.

These provenance requirements can be satisfied with our framework. First we construct a conceptual data model to represent this reality and then show how it could be mapped to PROV-DM. In figure 6.1 we present the model, which includes the concepts of process runs (for processes A and B), the tankers and tanks of intermediate and final product. The model also shows the relations between these.

In PROV-DM terms, both tankers and tanks are Entities and the run of processes A and B are Activities, which are concepts and relationships included in our model. The requirements are for operations of both backward and forward-tracing over the causal relationship between entities. The proposed implementation favours forward tracing, but an additional structure could be incorporated to facilitate backward tracing as well.

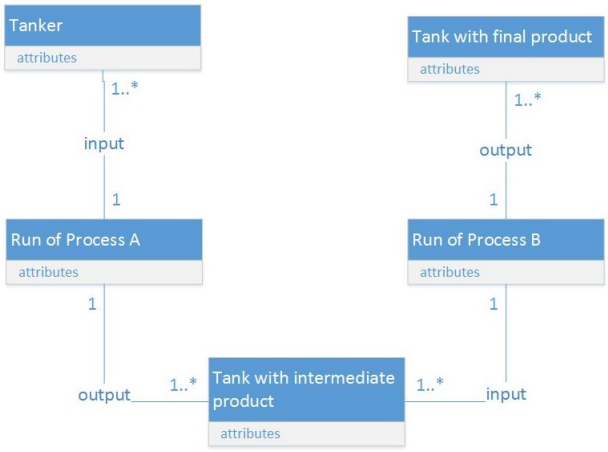

Figure 6.1: Conceptual data model for the industrial process.

6.2 Bibliographic references

This case study addresses the problem of extracting and representing academic publications and the references (citations) between them, in order to mine this data with different purposes. The bibliographic references' domain includes academic publications, their authors, the conferences where they were published and the editors of the conferences. Also, publications can have keywords associated to them.

With this information, we would like to identify all publications which were based on a given one, both directly and indirectly (i.e. publications which referenced another publication which in turn referenced the given one, or with more than one intermediate publication). Thus, it is possible to quantify the impact (both directly and indirectly) a publication has, and analyse it per year, per group of keywords, etc. Based on the impact of his publications, the impact of an author can be derived. These measures can be used to estimate the quality of the publications and authors. Analogously, the quality of conferences and editors can be assessed.

Here also, we start by representing the domain with a conceptual data model (figure 6.2), which is to be used to map to PROV-DM. First, a publication can reference many other publications, while at the same time, it may be referenced by many others. The publication is attributed to one or more authors and it is published in a conference edition. The conference edition has editors and belongs to a certain conference.

This problem is analogous to that of capturing and recording data provenance, and it can be represented using the PROV-DM model as follows: academic publications can be represented by Entities and authors by Agents, where the publication is attributed to the author. Each edition of a conference can be represented as a Collection Entity, which is composed of all its academic publications. The conference is attributed to the editors, which are also Agents. Finally, the editors act on behalf of the conference institution, another Agent.

This can be solved with the proposed framework because the different el-

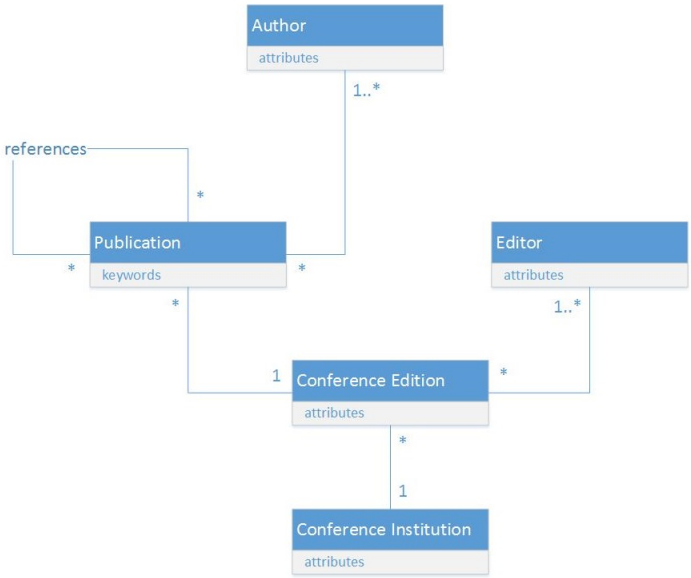

Figure 6.2: Conceptual data model for bibliographic references.

ements in the domain can be represented using the PROV-DM model that it uses. A specific module needs to be created for capturing "provenance" for this domain, which would have the responsibility of assigning identifiers to the publications and capturing the references between them, ideally retrieving the semantics of that reference, and finding the conference where it was published and its authors.

This information then needs to be sent to the provenance repository. The proposed repository does not provide support for PROV-DM Agents, so it needs to be extended to include them. The elicited requirements are for forward tracing operations, which are favoured by the proposed implementation. The mentioned extension should be done with this in mind.

6.3 Conclusions

We have seen two possible applications of the proposed provenance framework in scenarios that are very different from the traditional one of data processing, and shown how they could be solved with the proposed framework.

A key aspect for the adaptability of the framework is the selection of a general data model to represent provenance, PROV-DM, which has proven general enough to deal with the different case studies.

Chapter 7

Conclusions and Future Work

This thesis studies data provenance as a general problem and presents a framework for provenance capture and recording based on metadata and a conceptual data model, which can be used to offer provenance in a distributed environment with different technologies. In particular, it can be used in an environment where the nodes are, for example, a Hadoop system or a relational database.

Chapter 2 is dedicated to the survey of existing definitions of data provenance in different contexts. Its contributions are the unification of different terms used in the literature and the proposal of a comprehensive definition of data provenance.

In chapter 3 we present our case study and its requirements for provenance in a distributed environment with diverse technologies, in particular a Hadoop environment. We analyse existing solutions and conclude that none of them address all of aspects the scenario.

Chapter 4 is the analysis of the case study and the proposal of a framework to address it. The analysis consists of first constructing PCM, a conceptual data model for provenance, based on the requirements and data models of the surveyed works. One contribution of chapter 4 is the analysis of the challenges to satisfy the case study, in particular, when one of the nodes in the environment is a Hadoop system, analysing the challenges that can appear when the different tools of the Hadoop ecosystem interact with each other.

We close chapter 4 presenting the Framework for provenance capture and recording, the main contribution of this thesis. The framework defines roles and responsibilities to achieve data provenance capture and recording in this distributed scenario with diverse technologies, something not possible with the surveyed solutions. The framework is designed to be flexible to work with systems which can offer provenance both by a tracing procedure or via metadata, and in the case of offering it via metadata, that can record it both locally or centrally in the Provenance repository.

In chapter 5 we address the implementation of the framework. First we survey existing models for provenance interoperability, select OPM and PROV-DM, the two most widely used models, and assess them by comparing them to PCM. Then we define the services offered by our metadata repository (using PROV-DM terms) and give a detailed design of the Provenance repository using HBase and HDFS to be able to provide random access to large volumes of provenance metadata, based on the access requirements of our case study.

Then we present general aspects of the implementation of provenance, in particular those related to provenance capture. The contribution here is to show the relation between the characteristics of the systems offering provenance and the type of provenance offered, what to the best of our knowledge is not analysed by any work. We revisit the reviewed works analysing the characteristics of their implementations and what kinds of provenance they offer. We also make explicit some of the working hypothesis which are commonly implicit. We present a comparative summary of all the surveyed solutions.

We close chapter 5 showing how some of the surveyed solutions can be adapted to work within the framework acting in the provenance capture role and sending the provenance metadata to the Provenance repository, thus completing all the actors in our framework.

Since we have selected PROV-DM as the basis of the communication interfaces of the Provenance repository in order to offer a more general solution, in chapter 7 we show that the proposed framework can be used (with some extensions) to satisfy provenance requirements in different contexts outside of the traditional data processing scenarios.

Given that the proposed framework can manage provenance metadata from diverse sources and technologies and can accommodate large volumes of metadata, it can be used to provide provenance in a big data environment.

We have developed a provenance framework based on metadata, thus an interesting question that remains to be analysed is what is the possible size of the metadata relative to the size of the data. This will depend on the context of work and its characteristics, in particular, on the type of data transformations and the relation between inputs and outputs, and on the granularity of data items and their size. A starting point for this can be the works of Crawl et al. [44] and Ikeda et al. [61] who present different results in different situations. It could also be of interest to analyse how different characteristics of the context of work -for example the complexity of the data transformations- relate to the overhead of the provenance capture and the stress it implies for the provenance recording function.

Another line of work is to study how the provenance metadata could be compressed following the algorithms developed by Chapman et al. [40] while maintaining acceptable write and read response times. A possibility may be to run maintenance MapReduce jobs to perform these compression operations without penalizing write operations. In this case, it would be interesting to study the space gained against the performance penalty for read operations.

Also, we have proposed an implementation strategy based on Hadoop and HBase. It would be desirable to develop a set of guidelines for the tuning of this solution for it to cope with large volumes of metadata and situations of stress.

Another line for future work is that of analysing and developing management policies for the provenance metadata, which deal for example with historical metadata.

With our solution we have addressed the problem of offering data provenance in a distributed scenario, in particular we have shown how it could be applied for Hadoop. However, inside Hadoop we capture provenance only for MapReduce jobs. A natural next step is to extend the provenance capture provenance for data transformations occurring inside other tools of the Hadoop ecosystem, like Hive or Pig.

Bibliography

[1] OPM OWL Specification (Working draft), 2010 (accessed September 5, 2015). http://openprovenance.org/model/opmo/.

[2] OPM XML Schema Specification - Artifact, 2010 (accessed September 5, 2015). http://openprovenance.org/model/opmx#Artifact.

[3] OPM XML Schema Specification (Working draft), 2010 (accessed September 5, 2015). http://openprovenance.org/model/opmx/.

[4] Provenance Challenge, 2010 (accessed September 5, 2015). http://twiki.ipaw.info/bin/view/Challenge/.

[5] Dublin Core, 2011 (accessed September 5, 2015). http://wiki.dublincore.org/index.php/User_Guide.

[6] Provenance WG - Interoperability, 2011 (accessed September 5, 2015). http://www.w3.org/2011/prov/wiki/Interoperability/.

[7] W3C Provenance Incubator Group, 2011 (accessed September 5, 2015). http://www.w3.org/2005/Incubator/prov/.

[8] Dublin Core to PROV Mapping, 2013 (accessed September 5, 2015). http://www.w3.org/TR/prov-dc/.

[9] PROV-DM: The PROV Data Model, 2013 (accessed September 5, 2015). http://www.w3.org/TR/2013/REC-prov-dm-20130430.

[10] PROV-O: The PROV Ontology, 2013 (accessed September 5, 2015). http://www.w3.org/TR/2013/REC-prov-o-20130430/.

[11] PROV-XML: The PROV XML Schema, 2013 (accessed September 5, 2015). http://www.w3.org/TR/2013/NOTE-prov-xml-20130430/.

[12] Using Hive to interact with HBase, 2013 (accessed September 5, 2015). http://hortonworks.com/blog/hbase-via-hive-part-1/.

[13] Amazon Simple Storage Service (Amazon S3), (accessed September 5, 2015). https://aws.amazon.com/s3/.

[14] Apache Hadoop, (accessed September 5, 2015). https://hadoop.apache.org/.

[15] Apache HBase, (accessed September 5, 2015). http://hbase.apache.org/.

[16] Apache Hive, (accessed September 5, 2015). https://hive.apache.org/.

[17] Apache Mahout, (accessed September 5, 2015). http://mahout.apache.org/.

[18] Apache Oozie Workflow Scheduler for Hadoop, (accessed September 5, 2015). http://oozie.apache.org/.

[19] Apache Pig, (accessed September 5, 2015). http://pig.apache.org/.

[20] Apache Tajo, (accessed September 5, 2015). http://tajo.apache.org/.

[21] Apache Tez, (accessed September 5, 2015). https://tez.apache.org/.

[22] Azkaban Open-source Workflow Manager, (accessed September 5, 2015). https://azkaban.github.io/.

[23] Hadoop HDFS, (accessed September 5, 2015). https://hadoop.apache.org/docs/r1.2.1/hdfs_design.html/.

[24] Hadoop MapReduce, (accessed September 5, 2015). https://hadoop.apache.org/docs/r1.2.1/mapred_tutorial.html/.

[25] Impala, (accessed September 5, 2015). http://impala.io/.

[26] Taverna - Provenance management, (accessed September 5, 2015). http://www.taverna.org.uk/documentation/taverna-2-x/provenance/.

[27] Taverna Workflow Management System, (accessed September 5, 2015). http://www.taverna.org.uk/.

[28] The Kepler Project, (accessed September 5, 2015). https://kepler-project.org/.

[29] The OPM Provenance Model (OPM), (accessed September 5, 2015). http://openprovenance.org/.

[30] XML Schema 1.0 - xsd:ID, (accessed September 5, 2015). http://www.datypic.com/sc/xsd/t-xsd_ID.html.

[31] Sherif Akoush, Ripduman Sohan, and Andy Hopper. HadoopProv: Towards Provenance as a First Class Citizen in MapReduce. In *TaPP*, 2013.

[32] Ilkay Altintas, Oscar Barney, and Efrat Jaeger-Frank. Provenance collection support in the kepler scientific workflow system. In *Provenance and annotation of data*, pages 118–132. Springer, 2006.

[33] Yael Amsterdamer, Susan B Davidson, Daniel Deutch, Tova Milo, Julia Stoyanovich, and Val Tannen. Putting lipstick on pig: enabling database-style workflow provenance. *Proceedings of the VLDB Endowment*, 5(4):346–357, 2011.

[34] Brian Babcock, Shivnath Babu, Mayur Datar, Rajeev Motwani, and Jennifer Widom. Models and issues in data stream systems. In *Proceedings of the twenty-first ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, pages 1–16. ACM, 2002.

[35] Deepavali Bhagwat, Laura Chiticariu, Wang-Chiew Tan, and Gaurav Vijayvargiya. An annotation management system for relational databases. *The VLDB Journal*, 14(4):373–396, 2005.

[36] Rajendra Bose and James Frew. Lineage retrieval for scientific data processing: a survey. *ACM Computing Surveys (CSUR)*, 37(1):1–28, 2005.

[37] Peter Buneman, Sanjeev Khanna, and Tan Wang-Chiew. Why and where: A characterization of data provenance. In *Database Theoryï¿œICDT 2001*, pages 316–330. Springer, 2001.

[38] Lucian Carata, Sherif Akoush, Nikilesh Balakrishnan, Thomas Bytheway, Ripduman Sohan, Margo Selter, and Andy Hopper. A primer on provenance. *Communications of the ACM*, 57(5):52–60, 2014.

[39] Kno.e.sis Center. Provenir Ontology, 2009 (accessed September 5, 2015). `http://wiki.knoesis.org/index.php/Provenir_Ontology`.

[40] Adriane P Chapman, Hosagrahar V Jagadish, and Prakash Ramanan. Efficient provenance storage. In *Proceedings of the 2008 ACM SIGMOD international conference on Management of data*, pages 993–1006. ACM, 2008.

[41] James Cheney, Laura Chiticariu, and Wang-Chiew Tan. *Provenance in databases: Why, how, and where*. Now Publishers Inc, 2009.

[42] Paolo Ciccarese. swan-ontology, 2008 (accessed September 5, 2015). `https://code.google.com/p/swan-ontology/`.

[43] Paolo Ciccarese, Elizabeth Wu, Gwen Wong, Marco Ocana, June Kinoshita, Alan Ruttenberg, and Tim Clark. The swan biomedical discourse ontology. *Journal of biomedical informatics*, 41(5):739–751, 2008.

[44] Daniel Crawl, Jianwu Wang, and Ilkay Altintas. Provenance for MapReduce-based data-intensive workflows. In *Proceedings of the 6th workshop on Workflows in support of large-scale science*, pages 21–30. ACM, 2011.

[45] Yingwei Cui and Jennifer Widom. Practical lineage tracing in data warehouses. In *Data Engineering, 2000. Proceedings. 16th International Conference on*, pages 367–378. IEEE, 2000.

[46] Yingwei Cui and Jennifer Widom. Lineage tracing for general data warehouse transformations. *The VLDB Journalï¿œThe International Journal on Very Large Data Bases*, 12(1):41–58, 2003.

[47] Yingwei Cui, Jennifer Widom, and Janet L Wiener. Tracing the lineage of view data in a warehousing environment. *ACM Transactions on Database Systems (TODS)*, 25(2):179–227, 2000.

[48] Paulo Pinheiro Da Silva, Deborah L McGuinness, and Richard Fikes. A proof markup language for semantic web services. *Information Systems*, 31(4):381–395, 2006.

[49] Ramez Elmasri and Shamkant Navathe. *Fundamentals of database systems*. Pearson Education India, 2008.

[50] Deborah McGuinness et al. PML 3.0, 2013 (accessed September 5, 2015). `http://inference-web.org/wiki/PML_3.0`.

[51] Ian Foster, Jens Vockler, Michael Wilde, and Yong Zhao. Chimera: A virtual data system for representing, querying, and automating data derivation. In *Scientific and Statistical Database Management, 2002. Proceedings. 14th International Conference on*, pages 37–46. IEEE, 2002.

[52] Craig Franke, Samuel Morin, Artem Chebotko, John Abraham, and Pearl Brazier. Distributed semantic web data management in HBase and MySQL cluster. In *Cloud Computing (CLOUD), 2011 IEEE International Conference on*, pages 105–112. IEEE, 2011.

[53] James Frew and Rajendra Bose. Earth system science workbench: A data management infrastructure for earth science products. In *Scientific and Statistical Database Management, 2001. SSDBM 2001. Proceedings. Thirteenth International Conference on*, pages 180–189. IEEE, 2001.

[54] Lars George. *HBase: the definitive guide*. " O'Reilly Media, Inc.", 2011.

[55] Todd J Green, Grigoris Karvounarakis, and Val Tannen. Provenance semirings. In *Proceedings of the twenty-sixth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, pages 31–40. ACM, 2007.

[56] W3C Provenance Incubator Group. Provenance Vocabulary Mappings, 2010 (accessed September 5, 2015). `http://www.w3.org/2005/Incubator/prov/wiki/Provenance_Vocabulary_Mappings`.

[57] Olaf Hartig. Provenance Information in the Web of Data. In *LDOW*, 2009.

[58] Olaf Hartig. tRDF, 2013 (accessed September 5, 2015). `http://sourceforge.net/p/trdf/wiki/Home`.

[59] Apache HBase. Apache hbase reference guide. *Webpage available at http://wiki. apache. org/hadoop/Hbase/HbaseArchitecture. Webpage visited*, pages 04–04, 2012.

[60] Mohammad Rezwanul Huq, Andreas Wombacher, and Peter M G Apers. Inferring fine-grained data provenance in stream data processing: reduced storage cost, high accuracy. In *Database and Expert Systems Applications*, pages 118–127. Springer, 2011.

[61] Robert Ikeda, Hyunjung Park, and Jennifer Widom. Provenance for generalized map and reduce workflows. 2011.

[62] Phokion G Kolaitis. Schema mappings, data exchange, and metadata management. In *Proceedings of the twenty-fourth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, pages 61–75. ACM, 2005.

[63] Thomas Lee, Stéphane Bressan, and Stuart E Madnick. Source Attribution for Querying Against Semi-structured Documents. In *Workshop on Web Information and Data Management*, pages 33–39, 1998.

[64] Ling Liu and M Tamer Zsu. *Encyclopedia of database systems*. Springer Publishing Company, Incorporated, 2009.

[65] Arunprasad P Marathe. Tracing lineage of array data. *Journal of Intelligent Information Systems*, 17(2-3):193–214, 2001.

[66] Deborah L McGuinness, Li Ding, Paulo Pinheiro Da Silva, and Cynthia Chang. Pml 2: A modular explanation interlingua. In *ExaCt*, pages 49–55, 2007.

[67] Luc Moreau, Ben Clifford, Juliana Freire, Joe Futrelle, Yolanda Gil, Paul Groth, Natalia Kwasnikowska, Simon Miles, Paolo Missier, Jim Myers, and Others. The open provenance model core specification (v1. 1). *Future Generation Computer Systems*, 27(6):743–756, 2011.

[68] Luc Moreau and Paul Groth. Fourth Provenance Challenge, 2010 (accessed September 5, 2015). `http://twiki.ipaw.info/bin/view/Challenge/FourthProvenanceChallenge`.

[69] Luc Moreau, Paul Groth, Ivan Herman, and Sandro Hawke. W3C Provenance Working Group, 2013 (accessed September 5, 2015). `http://www.w3.org/2011/prov/wiki/Main_Page`.

[70] Luc Moreau and Paolo Missier. Prov-dm: The prov data model. 2013.

[71] Christopher Olston, Benjamin Reed, Utkarsh Srivastava, Ravi Kumar, and Andrew Tomkins. Pig latin: a not-so-foreign language for data processing. In *Proceedings of the 2008 ACM SIGMOD international conference on Management of data*, pages 1099–1110. ACM, 2008.

[72] Carmen Pancerella, John Hewson, Wendy Koegler, David Leahy, Michael Lee, Larry Rahn, Christine Yang, James D Myers, Brett Didier, Renata McCoy, and Others. Metadata in the collaboratory for multi-scale chemical science. In *International Conference on Dublin Core and Metadata Applications*, pages pp—-121, 2003.

[73] Hyunjung Park, Robert Ikeda, and Jennifer Widom. Ramp: A system for capturing and tracing provenance in mapreduce workflows. 2011.

[74] Erhard Rahm and Philip A Bernstein. A survey of approaches to automatic schema matching. *the VLDB Journal*, 10(4):334–350, 2001.

[75] Satya S Sahoo and Amit P Sheth. Provenir ontology: Towards a framework for escience provenance management. 2009.

[76] Yogesh L Simmhan, Beth Plale, and Dennis Gannon. A survey of data provenance in e-science. *ACM Sigmod Record*, 34(3):31–36, 2005.

[77] Yogesh L Simmhan, Beth Plale, and Dennis Gannon. A survey of data provenance techniques. *Computer Science Department, Indiana University, Bloomington IN*, 47405, 2005.

[78] Jianwu Wang, Daniel Crawl, and Ilkay Altintas. Kepler+ Hadoop: a general architecture facilitating data-intensive applications in scientific workflow systems. In *Proceedings of the 4th Workshop on Workflows in Support of Large-Scale Science*, page 12. ACM, 2009.

[79] Tom White. *Hadoop: The definitive guide.* " O'Reilly Media, Inc.", 2012.

[80] Allison Woodruff and Michael Stonebraker. Supporting fine-grained data lineage in a database visualization environment. In *Data Engineering, 1997. Proceedings. 13th International Conference on*, pages 91–102. IEEE, 1997.

Appendix A

Glossary

We present the different terms used in the reviewed literature and the definitions given by the authors.

Actor. The implementation of a specific function that needs to be performed [44].

Data object. A named entity which was consumed or produced by a derivation [51].

Derivation. An actual execution of a program [51].

Director. Directors specify what flows as tokens between actors; how the communication between the actors is achieved; when actors execute (a.k.a. fire) [44].

Experiment. An instance of a model [53].

Experiment step or processing step. On of the steps in an experiment, defined in the model [53].

Model. A specification of a data transformation modelled as a DAG (Directed Acyclic Graph) that can have many instances called experiments [53].

Module. A data transformation [33].

Module invocation. An execution of a data transformation [33].

Science object. Can be either data or a process [53].

Token. Communication between actors that contain both data and messages [44].

Transformation.

1. An executable program, of which it is known how to invoke it [51].
2. Any procedure that takes data sets as input and produces data sets as output, without distinguishing specifications from executions [45].

Transformation Sequence. A directed acyclic graph (DAG) of transformations, which would be equivalent to a workflow, without distinguishing specifications from executions [45].

Workflow.

1. A Directed Acyclic Graph (DAG) in which every node is annotated with a module identifier (name), and edges pass data between modules [33].
2. A linked set of components -the Actors- that may execute under different Models of Computations -the directors- [44].

Workflow execution. A sequence of module invocations [33].

Appendix B

Bibliographic Search

The selection of bibliographic references was done using Google Scholar, taking into account the abstracts of the works and the number of citations they received. Also, when reading a work which referenced related works which were considered relevant, these were included as well. What is more, additional works of the relevant authors were also searched for.

We present the searches performed in Google Scholar, showing the date in which it was done, the search terms used and other filter applied. For each search we present the results, assigning each result an identifier, and showing its bibtex and abstract, the number of citations it received, and an assessment about its relevance for our work with three possible results: it is considered relevant and must be read, it is considered relevant but too specific on a particular subject so it could be left for a later read, or alternatively it is not considered relevant.

Since reading a work or an author could lead to other works, we represent the results as a tree, where the works are vertices and the aforementioned relations are the connections. We present the results in a DFS (Depth First Search) fashion, while including comments explaining why additional references were considered.

B.1 Search 1

Search Id	1
Search date	2014 March
Site	scholar.google.com
Search terms	data provenance
Other	

Results

Id	Reference Description	Citations	Assessment

1.1 [76]	@article{simmhan2005survey, title={A survey of data provenance in e-science}, author={Simmhan, Yogesh L and Plale, Beth and Gannon, Dennis}, journal={ACM Sigmod Record}, year={2005}, publisher={ACM}}	715	Read
	Abstract: Data management is growing in complexity as large-scale applications take advantage of the loosely coupled resources brought together by grid middleware and by abundant storage capacity. Metadata describing the data products used in and generated by these applications is essential to disambiguate the data and enable reuse. Data provenance, one kind of metadata, pertains to the derivation history of a data product starting from its original sources. In this paper we create a taxonomy of data provenance characteristics and apply it to current research efforts in e-science, focusing primarily on scientific workflow approaches. The main aspect of our taxonomy categorizes provenance systems based on why they record provenance, what they describe, how they represent and store provenance, and ways to disseminate it. The survey culminates with an identification of open research problems in the field.		
	We retrieve some of the referenced articles by 1.1, which we identify as 1.1.*.		
1.1.1 [80]	@inproceedings{woodruff1997supporting, title={Supporting fine-grained data lineage in a database visualization environment}, author={Woodruff, Allison and Stonebraker, Michael}, booktitle={Data Engineering, 1997. Proceedings. 13th International Conference on}, year={1997}, organization={IEEE}}	216	Read

	Abstract: The lineage of a datum records its processing history. Because such information can be used to trace the source of anomalies and errors in processed data sets, it is valuable to users for a variety of applications including investigation of anomalies and debugging. Traditional data lineage approaches rely on metadata. However, metadata does not scale well to fine-grained lineage, especially in large data sets. For example, it is not feasible to store all of the information necessary to trace from a specific floating point value in a processed data set to a particular satellite image pixel in a source data set. In this paper, we propose a novel method to support fine-grained data lineage. Rather than relying on metadata, our approach lazily computes lineage using a limited amount of information about the processing operators and the base data. We introduce the notions of weak inversion and verification. While our system does not perfectly invert the data, it uses weak inversion and verification to provide a number of guarantees about the lineage it generates. We propose a design for the implementation of weak inversion and verification in an object-relational database management system.		
1.1.2 [45]	@inproceedings{cui2000practical, title={Practical lineage tracing in data warehouses}, author={Cui, Yingwei and Widom, Jennifer}, booktitle={Data Engineering, 2000. Proceedings. 16th International Conference on}, year={2000}, organization={IEEE}}	156	Read
	Abstract: We consider the view data lineage problem in a warehousing environment: For a given data item in a materialized warehouse view, we want to identify the set of source data items that produced the view item. We formalize the problem, and we present a lineage tracing algorithm for relational views with aggregation. Based on our tracing algorithm, we propose a number of schemes for storing auxiliary views that enable consistent and efficient lineage tracing in a multi-source data warehouse. We report on a performance study of the various schemes, identifying which schemes perform best in which settings. Based on our results, we have implemented a lineage tracing package in the WHIPS data warehousing system prototype at Stanford. With this package, users can select view tuples of interest, then efficiently "drill through" to examine the exact source tuples that produced the view tuples of interest.		
1.1.2.1 [46]	@article{cui2003lineage, title={Lineage tracing for general data warehouse transformations}, author={Cui, Yingwei and Widom, Jennifer}, journal={The VLDB Journal}, year={2003}, publisher={Springer-Verlag New York, Inc.}}	343	Read

	Abstract: Data warehousing systems integrate information from operational data sources into a central repository to enable analysis and mining of the integrated information. During the integration process, source data typically undergoes a series of transformations, which may vary from simple algebraic operations or aggregations to complex "data cleansing" procedures. In a warehousing environment, the data lineage problem is that of tracing warehouse data items back to the original source items from which they were derived. We formally define the lineage tracing problem in the presence of general data warehouse transformations, and we present algorithms for lineage tracing in this environment. Our tracing procedures take advantage of known structure or properties of transformations when present, but also work in the absence of such information. Our results can be used as the basis for a lineage tracing tool in a general warehousing setting, and also can guide the design of data warehouses that enable efficient lineage tracing.		
1.1.2.2 [47]	@article{cui2000tracing, title={Tracing the lineage of view data in a warehousing environment}, author={Cui, Yingwei and Widom, Jennifer and Wiener, Janet L}, journal={ACM Transactions on Database Systems (TODS)}, year={2000}, publisher={ACM}}	324	Read
	We consider the view data lineage problem in a warehousing en vironment. For a given data item in a materialized warehouse view, we want to identify the set of source data items that produced the view item. We formally define the lineage problem, develop lineage tracing algorithms for relational views with aggregation, and propose mechanisms for performing consistent lineage tracing in a multi-source data warehousing environment. Our results can form the basis of a tool that allows analysts to browse warehouse data, select view tuples of interest, then "drill-through" to examine the exact source tuples that produced the view tuples of interest.		
1.1.2.3	@article{widom2004trio, title={Trio: A system for integrated management of data, accuracy, and lineage}, author={Widom, J.}, year={2004}, publisher={Stanford InfoLab}}	539	Read
1.1.3	Reference 1.9		
1.1.4 [36]	@article{bose2005lineage, title={Lineage retrieval for scientific data processing: a survey}, author={Bose, R.; Frew, J.}, year={2005}, publisher={ACM}}	377	Read

	Abstract: Scientific research relies as much on the dissemination and exchange of data sets as on the publication of conclusions. Accurately tracking the lineage (origin and subsequent processing history) of scientific data sets is thus imperative for the complete documentation of scientific work. Researchers are effectively prevented from determining, preserving, or providing the lineage of the computational data products they use and create, however, because of the lack of a definitive model for lineage retrieval and a poor fit between current data management tools and scientific software. Based on a comprehensive survey of lineage research and previous prototypes, we present a metamodel to help identify and assess the basic components of systems that provide lineage retrieval for scientific data products.		
1.1.5 [51]	@inproceedings{foster2002chimera, title={Chimera: A virtual data system for representing, querying, and automating data derivation}, author={Foster, I.; Vockler, J.; et al., year={2002}, organization={IEEE}}	671	Read
	Abstract: Much scientific data is not obtained from measurements but rather derived from other data by the application of computational procedures. We hypothesize that explicit representation of these procedures can enable documentation of data provenance, discovery of available methods, and on-demand data generation (so-called "virtual data"). To explore this idea, we have developed the Chimera virtual data system, which combines a virtual data catalog, for representing data derivation procedures and derived data, with a virtual data language interpreter that translates user requests into data definition and query operations on the database. We couple the Chimera system with distributed "Data Grid" services to enable on-demand execution of computation schedules constructed from database queries. We have applied this system to two challenge problems, the reconstruction of simulated collision event data from a high-energy physics experiment, and the search of digital sky survey data for galactic clusters, with promising results.		
1.1.6	@incollection{zhao2004semantically, title={Semantically linking and browsing provenance logs for e-science}, author={Zhao, Jun and Goble, Carole and Stevens, Robert and Bechhofer, Sean}, year={2004}, publisher={Springer}}	79	Specific. Log processing.

	Abstract: e-Science experiments are those performed using computer-based resources such as database searches, simulations or other applications. Like their laboratory based counterparts, the data associated with an e-Science experiment are of reduced value if other scientists are not able to identify the origin, or provenance, of those data. Provenance is the term given to metadata about experiment processes, the derivation paths of data, and the sources and quality of experimental components, which includes the scientists themselves, related literature, etc. Consequently provenance metadata are valuable resources for e-Scientists to repeat experiments, track versions of data and experiment runs, verify experiment results, and as a source of experimental insight. One specific kind of in silico experiment is a workflow. In this paper we describe how we can assemble a SemanticWeb of workflow provenance logs that allows a bioinformatician to browse and navigate between experimental components by generating hyperlinks based on semantic annotations associated with them. By associating well-formalized semantics with workflow logs we take a step towards integration of process provenance information and improved knowledge discovery.		
1.1.7 [53]	@inproceedings{frew2001earth, title={Earth system science workbench: A data management infrastructure for earth science products}, author={Frew, J.; Bose, R.}, year={2001}, organization={IEEE}}	125	Read
	Abstract: The Earth System Science Workbench (ESSW) is a nonintrusive data management infrastructure for researchers who must also be data publishers. An implementation of ESSW to track the processing of locally received satellite images is presented, demonstrating the Workbench's transparent and robust support for archiving and publishing data products. ESSW features a Lab Notebook metadata service, a No Duplicate- Write Once Read Many (ND-WORM) storage service, and Web user interface tools. The Lab Notebook logs processes (experiments) and their relationships via a custom APl to XML documents stored in a relational database. The ND-WORM provides a managed storage archive for the Lab Notebook by keeping unique file digests and namespace metadata, also in a relational database. ESSW Notebook tools allow product searching and ordering, and file and metadata management.		
1.2 [37]	@incollection{buneman2001and, title={Why and where: A characterization of data provenance}, author={Buneman, P.; Khanna, S.; Wang-Chiew, T.}, booktitle={Database Theory—ICDT 2001}, year={2001}, publisher={Springer}}	841	Read

	Abstract: With the proliferation of database views and curated databases, the issue of data provenance - where a piece of data came from and the process by which it arrived in the database - is becoming increasingly important, especially in scientific databases where understanding provenance is crucial to the accuracy and currency of data. In this paper we describe an approach to computing provenance when the data of interest has been created by a database query. We adopt a syntactic approach and present results for a general data model that applies to relational databases as well as to hierarchical data such as XML. A novel aspect of our work is a distinction between "why" provenance (refers to the source data that had some influence on the existence of the data) and "where" provenance (refers to the location(s) in the source databases from which the data was extracted).		
1.2.1	Reference 1.1.1		
1.2.2	Reference 1.1.2		
1.2.3 [35]	@article{bhagwat2005annotation, title={An annotation management system for relational databases}, author={Bhagwat, D.; Chiticariu, L. et al.}, journal={The VLDB Journal}, year={2005}}	253	Read
	Abstract: We present an annotationmanagement system for relational databases. In this system, every piece of data in a relation is assumed to have zero or more annotations associated with it and annotations are propagated along, from the source to the output, as data is being transformed through a query. Such an annotation management system could be used for understanding the provenance (aka lin- eage) of data, who has seen or edited a piece of data or the quality of data, which are useful functionalities for applica- tions that deal with integration of scientific and biological data. We present an extension, pSQL, of a fragment of SQL that has three different types of annotation propagation schemes, each useful for different purposes. The default scheme propagates annotations according to where data is copied from. The default-all scheme propagates annotations according to where data is copied from among all equiv- alent formulations of a given query. The custom scheme al- lows a user to specify howannotations should propagate.We present a storage scheme for the annotations and describe algorithms for translating a pSQL query under each propagation scheme into one or more SQL queries that would correctly retrieve the relevant annotations according to the specified propagation scheme. For the default-all scheme, we also showhowwe generate finitely many queries that can simulate the annotation propagation behavior of the set of all equivalent queries,which is possibly infinite. The algorithms are implemented and the feasibility of the system is demon- strated by a set of experiments that we have conducted.		

1.2.4 [72]	@inproceedings{pancerella2003metadata, title={Metadata in the collaboratory for multi-scale chemical science}, author={Pancerella, C.; Hewson, J.; et al.}, booktitle={International Conference on Dublin Core and Metadata Applications}, year={2003}}	57	Read
	Abstract: The goal of the Collaboratory for the Multi-scale Chemical Sciences (CMCS) [1] is to develop an informatics-based approach to synthesizing multi-scale chemistry information to create knowledge in the chemical sciences. CMCS is using a portal and metadata-aware content store as a base for building a system to support inter-domain knowledge exchange in chemical science. Key aspects of the system include configurable metadata extraction and translation, a core schema for scientific pedigree, and a suite of tools for managing data and metadata and visualizing pedigree relationships between data entries. CMCS metadata is represented using Dublin Core with metadata extensions that are useful to both the chemical science community and the science community in general. CMCS is working with several chemistry groups who are using the system to collaboratively assemble and analyze existing data to derive new chemical knowledge. In this paper we discuss the project's metadata-related requirements, the relevant software infrastructure, core metadata schema, and tools that use the metadata to enhance science.		
1.3	@incollection{buneman2000data, title={Data provenance: Some basic issues}, author={Buneman, P.; Khanna, S.; Tan, Wang-Chiew}, booktitle={FST TCS 2000: Foundations of Software Technology and Theoretical Computer Science}, year={2000}, publisher={Springer}}	183	Specific. Data from the web.
	Abstract: The ease with which one can copy and transform data on the Web, has made it increasingly dicult to determine the origins of a piece of data. We use the term data provenance to refer to the process of tracing and recording the origins of data and its movement between databases. Provenance is now an acute issue in scientic databases where it central to the validation of data. In this paper we discuss some of the technical issues that have emerged in an initial exploration of the topic		
1.4	@article{moreau2008provenance, title={The provenance of electronic data}, author={Moreau, L.; Groth, P.; Miles, S.;Vazquez-Salceda, J.; and others}, journal={Communications of the ACM}, year={2008}, publisher={ACM}}	141	Specific. Provenance life cycle.

	Abstract: In the study of fine art, provenance refers to the documented history of some art object. Given that documented history, the object attains an authority that allows scholars to appreciate its importance with respect to other works, whereas, in the absence of such history, the object may be treated with some skepticism. Our IT landscape is evolving as illustrated by applications that are open, composed dynamically, and that discover results and services on the fly. Against this challenging background, it is crucial for users to be able to have confidence in the results produced by such applications. If the provenance of data produced by computer systems could be determined as it can for some works of art, then users, in their daily applications, would be able to interpret and judge the quality of data better. We introduce a provenance lifecycle and advocate an open approach based on two key principles to support a notion of provenance in computer systems: documentation of execution and user-tailored provenance queries.		
1.5 [57]	@inproceedings{hartig2009provenance, title={Provenance Information in the Web of Data.}, author={Hartig, Olaf}, booktitle={LDOW}, year={2009}}	128	Specific. Linked data and data usage.
	Abstract: The openness of the Web and the ease to combine linked data from different sources creates new challenges. Systems that consume linked data must evaluate quality and trustworthiness of the data. A common approach for data quality assessment is the analysis of provenance information. For this reason, this paper discusses provenance of data on the Web and proposes a suitable provenance model. While traditional provenance research usually addresses the creation of data, our provenance model also represents data access, a dimension of provenance that is particularly relevant in the context of Web data. Based on our model we identify options to obtain provenance information and we raise open questions concerning the publication of provenance-related metadata for linked data on the Web.		
1.6	@incollection{szomszor2003recording, title={Recording and reasoning over data provenance in web and grid services}, author={Szomszor, Martin and Moreau, Luc}, booktitle={On the move to meaningful Internet systems 2003: CoopIS, DOA, and ODBASE}, year={2003}, publisher={Springer}}	120	Specific. Grid and Web Services.

	Abstract: Large-scale, dynamic and open environments such as the Grid and Web Services build upon existing computing infrastructures to supply dependable and consistent large-scale computational systems. This kind of architecture has been adopted by those working with business and scientific information systems allowing them to exploit extensive and diverse computing resources to perform complex data processing tasks. In such systems, results are often derived by composing multiple, geographically distributed, heterogeneous services as specified by intricate workflow management. This leads to the undesirable situation where the results are known, but the means by which they were achieved is not. With both scientific experiments and business transactions, the notion of lineage and dataset derivation is of paramount importance since without it, information is potentially worthless. We address the issue of data provenance, the description of the origin of a piece of data, in these environments showing the requirements, uses and implementation difficulties. We propose an infrastructure level support for a provenance recording capability for service-oriented architectures such as the Grid and Web Services. We also offer services to view and retrieve provenance and we provide a mechanism by which provenance is used to determine whether previous computed results are still up to date.		
1.7	@inproceedings{simmhan2006framework, title={A framework for collecting provenance in data-centric scientific workflows}, author={Simmhan, Yogesh L and Plale, Beth and Gannon, Dennis}, booktitle={Web Services, 2006. ICWS'06. International Conference on}, year={2006}, organization={IEEE}}	112	Specific.
	Abstract: As wireless networking becomes more robust and sensor technology, cameras, and low-power compute devices reach the consumer market as inexpensive commodity products, the proliferation of relatively inexpensive specialized sensors and networks will have an impact on scientific computing that parallels that of the commercial space. Specifically, the increasing ability to sense the world around us will result in a growing need for scientific data-driven applications that are under the control of data-centric workflows. These workflows are complex, nondeterministic, and event-driven. The focus of our work is on provenance collection for data-centric workflows. The challenge we address is to record uniform and usable provenance metadata that meets the domain needs while minimizing the modification burden on the service authors and the performance overhead on the workflow engine and the services. The framework, which is based on a loosely-coupled publish-subscribe architecture for propagating provenance activities, is argued on the grounds that it reduces programmer overhead and can satisfy the needs of detailed provenance collection with minimal performance overhead (in the range of 1%).		

1.8	@incollection{bowers2006model, title={A model for user-oriented data provenance in pipelined scientific workflows}, author={Bowers, S.; McPhillips, T.; et al.}, booktitle={Provenance and Annotation of Data}, year={2006}, publisher={Springer}}	104	Specific.
	Abstract: Integrated provenance support promises to be a chief advantage of scientific workflow systems over scriptbased alternatives. While it is often recognized that information gathered during scientific workflow execution can be used automatically to increase fault tolerance (via checkpointing) and to optimize performance (by reusing intermediate data products in future runs), it is perhaps more significant that provenance information may also be used by scientists to reproduce results from earlier runs, to explain unexpected results, and to prepare results for publication. Current workflow systems offer little or no direct support for these "scientistoriented" queries of provenance information. Indeed the use of advanced execution models in scientific workflows (e.g., process networks, which exhibit pipeline parallelism over streaming data) and failure to record certain fundamental events such as state resets of processes, can render existing provenance schemas useless for scientific applications of provenance. We develop a simple provenance model that is capable of supporting a wide range of scientific use cases even for complex models of computation such as process networks. Our approach reduces these use cases to database queries over event logs, and is capable of reconstructing complete data and invocation dependency graphs for a workflow run.		
1.9 [77]	@article{simmhan2005survey-techniques, title={A survey of data provenance techniques}, author={Simmhan, Yogesh L and Plale, Beth and Gannon, Dennis}, journal={Computer Science Department, Indiana University, Bloomington IN}, volume={47405}, year={2005}}	94	Read

	Abstract: Data management is growing in complexity as large-scale applications take advantage of the loosely coupled resources brought together by grid middleware and by abundant storage capacity. Metadata describing the data products used in and generated by these applications is essential to disambiguate the data and enable reuse. Data provenance, one kind of metadata, pertains to the derivation history of a data product starting from its original sources. The provenance of data products generated by complex transformations such as workflows is of considerable value to scientists. From it, one can ascertain the quality of the data based on its ancestral data and derivations, track back sources of errors, allow automated re-enactment of derivations to update a data, and provide attribution of data sources. Provenance is also essential to the business domain where it can be used to drill down to the source of data in a data warehouse, track the creation of intellectual property, and provide an audit trail for regulatory purposes. In this paper we create a taxonomy of data provenance techniques, and apply the classification to current research efforts in the field. The main aspect of our taxonomy categorizes provenance systems based on why they record provenance, what they describe, how they represent and store provenance, and ways to disseminate it. Our synthesis can help those building scientific and business metadata-management systems to understand existing provenance system designs. The survey culminates with an identification of open research problems in the field.		
1.10	@article{clifford2008tracking, title={Tracking provenance in a virtual data grid}, author={Clifford, B.; Foster, I.; Voeckler, J.; et al.}, journal={Concurrency and Computation: Practice and Experience}, year={2008}, publisher={Wiley Online Library}}	73	Specific. Virtual data grids.
	Abstract: The virtual data system (VDS) was developed within the Grid Physics Network (GriPhyN) project with the goal of allowing data sets to be described prior to, and separately from, their physical materialization. A virtual data language (VDL) is used to describe how data files are computed from input data files and parameters. A runtime system provides for the on-demand derivation of files in a variety of execution environments, including distributed Grids. A virtual data catalog (VDC) stores descriptions of both virtual data sets and computational procedures, together with logs of process executions and metadata annotations on both procedures and data, provided by users or applications. This integration of three types of provenance data—program structure, runtime logs, and annotation—into a unified relational schema enables powerful discovery and analysis operations.		

B.2 Search 2

Search Id	2
Search date	2014 August
Site	scholar.google.com
Search terms	data provenance map reduce
Other	since 2010

Results

Id	Reference Description	Citations	Assessment
2.1	@inproceedings{zaharia2010spark, title={Spark: cluster computing with working sets}, author={Zaharia, Matei and Chowdhury, Mosharaf and Franklin, Michael J and Shenker, Scott and Stoica, Ion}, booktitle={Proceedings of the 2nd USENIX conference on Hot topics in cloud computing}, pages={10–10}, year={2010}}	320	Not to read
	Abstract: MapReduce and its variants have been highly successful in implementing large-scale data-intensive applications on commodity clusters. However, most of these systems are built around an acyclic data flow model that is not suitable for other popular applications. This paper focuses on one such class of applications: those that reuse a working set of data across multiple parallel operations. This includes many iterative machine learning algorithms, as well as interactive data analysis tools. We propose a new framework called Spark that supports these applications hile retaining the scalability and fault tolerance of MapReduce. To achieve these goals, Spark introduces an abstraction called resilient distributed datasets (RDDs). An RDD is a read-only collection of objects partitioned across a set of machines that can be rebuilt if a partition is lost. Spark can outperform Hadoop by 10x in iterative machine learning jobs, and can be used to interactively query a 39 GB dataset with sub-second response time.		
2.2 [61]	@article{ikeda2011provenance, title={Provenance for generalized map and reduce workflows}, author={Ikeda, Robert and Park, Hyunjung and Widom, Jennifer}, year={2011}, publisher={Stanford InfoLab}}	29	Read

	Abstract: We consider a class of workflows, which we call generalized map and reduce workflows (GMRWs), where input data sets are processed by an acyclic graph of map and reduce functions to produce output results. We show how data provenance (also sometimes called lineage) can be captured for map and reduce functions transparently. The captured provenance can then be used to support backward tracing (finding the input subsets that contributed to a given output element) and forward tracing (determining which output elements were derived from a particular input element). We provide formal underpinnings for provenance in GMRWs, and we identify properties that are guaranteed to hold when provenance is applied recursively. We have built a prototype system that supports provenance capture and tracing as an extension to Hadoop. Our system uses a wrapper-based approach, requiring little if any user intervention in most cases, and retaining Hadoop's parallel execution and fault tolerance. Performance numbers from our system are reported.		
2.3	@inproceedings{olston2011inspector, title={Inspector Gadget: A framework for custom monitoring and debugging of distributed dataflows}, author={Olston, C.; Reed, B.}, booktitle={Proceedings of the 2011 ACM SIGMOD International Conference on Management of data}, year={2011}, organization={ACM}}	13	Specific. Framework for debugging.
	Abstract: We consider how to monitor and debug query processing dataflows, in distributed environments such as Pig/Hadoop. Our work is motivated by a series of informal user interviews, which revealed that monitoring and debugging needs are both pressing and diverse. In response to these interviews, we created a framework for custom dataflow instrumentation, called Inspector Gadget (IG). IG makes it easy to write a wide variety of monitoring and debugging behaviors, and attaches seamlessly to an existing, unmodified dataflow environment such as Pig. We have implemented a dozen user-requested tools in Inspector Gadget, each in just a few hundred lines of Java code. The performance overhead is modest in most cases. Our Pig-based implementation of IG, called Penny, is slated for public release in mid-2011, in conjunction with the upcoming Apache Pig v0.9 release.		

2.4	@inproceedings{sarma2013upper, title={Upper and lower bounds on the cost of a map-reduce computation}, author={Sarma, Anish Das and Afrati, Foto N and Salihoglu, Semih and Ullman, Jeffrey D}, booktitle={Proceedings of the VLDB Endowment}, volume={6}, number={4}, pages={277–288}, year={2013}, organization={VLDB Endowment}}	19	Not to read
	Abstract: In this paper we study the tradeoff between parallelism and communication cost in a map-reduce computation. For any problem that is not "embarrassingly parallel," the finer we partition the work of the reducers so that more parallelism can be extracted, the greater will be the total communication between mappers and reducers. We introduce a model of problems that can be solved in a single round of mapreduce computation. This model enables a generic recipe for discovering lower bounds on communication cost as a function of the maximum number of inputs that can be assigned to one reducer. We use the model to analyze the tradeoff for three problems: finding pairs of strings at Hamming distance d, finding triangles and other patterns in a larger graph, and matrix multiplication. For finding strings of Hamming distance 1, we have upper and lower bounds that match exactly. For triangles and many other graphs, we have upper and lower bounds that are the same to within a constant factor. For the problem of matrix multiplication, we have matching upper and lower bounds for one-round map-reduce algorithms. We are also able to explore tworound map-reduce algorithms for matrix multiplication and show that these never have more communication, for a given reducer size, than the best one-round algorithm, and often have significantly less.		
2.5 [60]	@inproceedings{huq2011inferring, title={Inferring fine-grained data provenance in stream data processing: reduced storage cost, high accuracy}, author={Huq, M.; Wombacher, A.; Apers, P.}, booktitle={Database and Expert Systems Applications}, year={2011}, organization={Springer}}	12	Read

	Abstract: Fine-grained data provenance ensures reproducibility of results in decision making, process control and e-science applications. However, maintaining this provenance is challenging in stream data processing because of its massive storage consumption, especially with large overlapping sliding windows. In this paper, we propose an approach to infer fine-grained data provenance by using a temporal data model and coarse-grained data provenance of the processing. The approach has been evaluated on a real dataset and the result shows that our proposed inferring method provides provenance information as accurate as explicit fine-grained provenance at reduced storage consumption.		
2.5.1	@inproceedings{chapman2008efficient, title={Efficient provenance storage}, author={Chapman, Adriane P and Jagadish, Hosagrahar V and Ramanan, Prakash}, booktitle={Proceedings of the 2008 ACM SIGMOD international conference on Management of data}, pages={993–1006}, year={2008}, organization={ACM} }	142	Specific. Provenance storage.
	Abstract: As the world is increasingly networked and digitized, the data we store has more and more frequently been chopped, baked, diced and stewed. In consequence, there is an increasing need to store and manage provenance for each data item stored in a database, describing exactly where it came from, and what manipulations have been applied to it. Storage of the complete provenance of each data item can become prohibitively expensive. In this paper, we identify important properties of provenance that can be used to considerably reduce the amount of storage required. We identify three different techniques: a family of factorization processes and two methods based on inheritance, to decrease the amount of storage required for provenance. We have used the techniques described in this work to significantly reduce the provenance storage costs associated with constructing MiMI, a warehouse of data regarding protein interactions, as well as two provenance stores, Karma and PReServ, produced through workflow execution. In these real provenance sets, we were able to reduce the size of the provenance by up to a factor of 20. Additionally, we show that this reduced store can be queried efficiently and further that incremental changes can be made inexpensively		
2.6 [73]	@article{park2011ramp, title={Ramp: A system for capturing and tracing provenance in mapreduce workflows}, author={Park, Hyunjung and Ikeda, Robert and Widom, Jennifer}, year={2011}, publisher={Stanford InfoLab}}	16	Read

	Abstract: RAMP (Reduce And Map Provenance) is an extension to Hadoop that supports provenance capture and tracing for workflows of MapReduce jobs. RAMP uses a wrapper-based approach, requiring little if any user intervention in most cases, while retaining Hadoop's parallel execution and fault tolerance. We demonstrate RAMP on a real-world MapReduce workflow generated from a Pig script that performs sentiment analysis over Twitter data. We show how RAMP's automatic provenance capture and tracing capabilities provide a convenient and efficient means of drilling-down and verifying output elements.		
2.7	@inproceedings{olston2011nova, title={Nova: continuous pig/hadoop workflows}, author={Olston, C.; Chiou, G.; et al.}, booktitle={Proceedings of the 2011 ACM SIGMOD International Conference on Management of data}, year={2011}, organization={ACM}}	55	Not to read
	Abstract: This paper describes a workflow manager developed and deployed at Yahoo called Nova, which pushes continually-arriving data through graphs of Pig programs executing on Hadoop clusters. (Pig is a structured dataflow language and runtime for the Hadoop map-reduce system.) Nova is like data stream managers in its support for stateful incremental processing, but unlike them in that it deals with data in large batches using disk-based processing. Batched incremental processing is a good fit for a large fraction of Yahoo's data processing use-cases, which deal with continually-arriving data and benefit from incremental algorithms, but do not require ultra-low-latency processing.		
2.8	@inproceedings{olston2011ibis, title={Ibis: A Provenance Manager for Multi-Layer Systems.}, author={Olston, Christopher and Sarma, Anish Das}, booktitle={CIDR}, pages={152–159}, year={2011} }	8	Read

	Abstract: End-to-end data processing environments are often comprised of several independently-developed (sub-)systems, e.g. for engineering, organizational or historical reasons. Unfortunately this situation harms usability. For one thing, systems created independently tend to have disparate capabilities in terms of what metadata is retained and how it can be queried. If something goes wrong it can be very difficult to trace execution histories across the various sub-systems. One solution is to ship each sub-system's metadata to a central metadata manager that integrates it and offers a powerful and uniform query interface. This paper describes a metadata manager we are building, called Ibis. Perhaps the greatest challenge in this context is dealing with data provenance queries in the presence of mixed granularities of metadata—e.g. rows vs. column groups vs. tables; mapreduce job slices vs. relational operators—supplied by different sub-systems. The central contribution of our work is a formal model of multi-granularity data provenance relationships, and a corresponding query language. We illustrate the simplicity and power of our query language via several real-world-inspired examples. We have implemented all of the functionality described in this paper.		
2.9	@inproceedings{zhao2011opportunities, title={Opportunities and challenges in running scientific workflows on the cloud}, author={Zhao, Yong and Fei, Xubo and Raicu, Ioan and Lu, Shiyong}, booktitle={Cyber-Enabled Distributed Computing and Knowledge Discovery (CyberC), 2011 International Conference on}, pages={455–462}, year={2011}, organization={IEEE}}	36	Not to read
	Abstract: Cloud computing is gaining tremendous momentum in both academia and industry. The application of Cloud computing, however, has mostly focused on Web applications and business applications; while the recognition of using Cloud computing to support large-scale workflows, especially dataintensive scientific workflows on the Cloud is still largely overlooked. We coin the term "Cloud Workflow", to refer to the specification, execution, provenance tracking of large-scale scientific workflows, as well as the management of data and computing resources to enable the execution of scientific workflows on the Cloud. In this paper, we analyze why there has been such a gap between the two technologies, and what it means to bring Cloud and workflow together; we then present the key challenges in running Cloud workflow, and discuss the research opportunities in realizing workflows on the Cloud.		

2.10 [33]	@article{amsterdamer2011putting, title={Putting lipstick on pig: enabling database-style workflow provenance}, author={Amsterdamer, Y.; Davidson, S.; et al.}, year={2011}, publisher={VLDB Endowment}}	35	Read
	Abstract: Workflow provenance typically assumes that each module is a "black-box", so that each output depends on all inputs (coarse-grained dependencies). Furthermore, it does not model the internal state of a module, which can change between repeated executions. In practice, however, an output may depend on only a small subset of the inputs (fine-grained dependencies) as well as on the internal state of the module. We present a novel provenance framework that marries database-style and workflow-style provenance, by using Pig Latin to expose the functionality of modules, thus capturing internal state and fine-grained dependencies. A critical ingredient in our solution is the use of a novel form of provenance graph that models module invocations and yields a compact representation of fine-grained workflow provenance. It also enables a number of novel graph transformation operations, allowing to choose the desired level of granularity in provenance querying (ZoomIn and ZoomOut), and supporting "what-if" workflow analytic queries. We implemented our approach in the Lipstick system and developed a benchmark in support of a systematic performance evaluation. Our results demonstrate the feasibility of tracking and querying fine-grained workflow provenance.		
2.10.1	Reference 1.2		
2.10.2 [41]	@book{cheney2009provenance, title={Provenance in databases: Why, how, and where}, author={Cheney, James and Chiticariu, Laura and Tan, Wang-Chiew}, year={2009}, publisher={Now Publishers Inc} }	289	Read
	Abstract: Different notions of provenance for database queries have been proposed and studied in the past few years. In this article, we detail three main notions of database provenance, some of their applications, and compare and contrast amongst them. Specifically, we review why, how, and where provenance, describe the relationships among these notions of provenance, and describe some of their applications in confidence computation, view maintenance and update, debugging, and annotation propagation.		
2.10.3	Reference 2.2		

2.10.4 [55]	@inproceedings{green2007provenance, title={Provenance semirings}, author={Green, Todd J and Karvounarakis, Grigoris and Tannen, Val}, booktitle={Proceedings of the twenty-sixth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems}, pages={31–40}, year={2007}, organization={ACM} }	430	Read
	We show that relational algebra calculations for incomplete databases, probabilistic databases, bag semantics and why provenance are particular cases of the same general algorithms involving semirings. This further suggests a comprehensive provenance representation that uses semirings of polynomials. We extend these considerations to datalog and semirings of formal power series. We give algorithms for datalog provenance calculation as well as datalog evaluation for incomplete and probabilistic databases. Finally, we show that for some semirings containment of conjunctive queries is the same as for standard set semantics.		
2.11 [44]	@inproceedings{crawl2011provenance, title={Provenance for mapreduce-based data-intensive workflows}, author={Crawl, Daniel and Wang, Jianwu and Altintas, Ilkay}, booktitle={Proceedings of the 6th workshop on Workflows in support of large-scale science}, pages={21–30}, year={2011}, organization={ACM}}		Read
	Abstract: MapReduce has been widely adopted by many business and scientific applications for data-intensive processing of large datasets. There are increasing efforts for workflows and systems to work with the MapReduce programming model and the Hadoop environment including our work on a higher- level programming model for MapReduce within the Kepler Scientific Workflow System. However, to date, provenance of MapReduce-based workflows and its effects on workflow execution performance have not been studied in depth. In this paper, we present an extension to our earlier work on MapReduce in Kepler to record the provenance of MapReduce workflows created using the Kepler+Hadoop framework. In particular, we present: (i) adatamodelthat is able to capture provenance inside a MapReduce job as well as the provenance for the workflow that submitted it; (ii) an extension to the Kepler+Hadoop architecture to record provenance using this data model on MySQL Cluster; (iii) a programming interface to query the collected information; and (iv) an evaluation of the scalability of collecting and querying this provenance information using two scenarios with different characteristics.		

2.11.1 [52]	@inproceedings{franke2011distributed, title={Distributed semantic web data management in HBase and MySQL cluster}, author={Franke, Craig and Morin, Samuel and Chebotko, Artem and Abraham, John and Brazier, Pearl}, booktitle={Cloud Computing (CLOUD), 2011 IEEE International Conference on}, pages={105–112}, year={2011}, organization={IEEE}	40	Specific.
	Abstract: Various computing and data resources on the Web are being enhanced with machine-interpretable semantic descriptions to facilitate better search, discovery and integration. This interconnected metadata constitutes the Semantic Web, whose volume can potentially grow the scale of the Web. Efficient management of Semantic Web data, expressed using the W3C's Resource Description Framework (RDF), is crucial for supporting new data-intensive, semantics-enabled applications. In this work, we study and compare two approaches to distributed RDF data management based on emerging cloud computing technologies and traditional relational database clustering technologies. In particular, we design distributed RDF data storage and querying schemes for HBase and MySQL Cluster and conduct an empirical comparison of these approaches on a cluster of commodity machines using datasets and queries from the Third Provenance Challenge and Lehigh University Benchmark. Our study reveals interesting patterns in query evaluation, shows that our algorithms are promising, and suggests that cloud computing has a great potential for scalable Semantic Web data management		
2.11.2 [32]	@incollection{altintas2006provenance, title={Provenance collection support in the kepler scientific workflow system}, author={Altintas, Ilkay and Barney, Oscar and Jaeger-Frank, Efrat}, booktitle={Provenance and annotation of data}, pages={118–132}, year={2006}, publisher={Springer} }	225	Read

	Abstract: In many data-driven applications, analysis needs to be performed on scientific information obtained from several sources and generated by computations on distributed resources. Systematic analysis of this scientific infor- mation unleashes a growing need for automated data-driven applications that also can keep track of the provenance of the data and processes with little user interaction and overhead. Such data analysis can be facilitated by the recent ad- vancements in scientific workflow systems. A major profit when using scientific workflow systems is the ability to make provenance collection a part of the workflow. Specifically, provenance should include not only the standard data lineage information but also information about the context in which the workflow was used, execution that processed the data, and the evolution of the workflow design. In this paper we describe a complete framework for data and process provenance in the Kepler Scientific Workflow System. We outline the requirements and issues related to data and workflow provenance in a multi- disciplinary workflow system and introduce how generic provenance capture can be facilitated in Kepler's actor-oriented workflow environment. We also describe the usage of the stored provenance information for efficient rerun of scientific workflows.		
2.11.3 [78]	@inproceedings{wang2009kepler+, title={Kepler+ Hadoop: a general architecture facilitating data-intensive applications in scientific workflow systems}, author={Wang, Jianwu and Crawl, Daniel and Altintas, Ilkay}, booktitle={Proceedings of the 4th Workshop on Workflows in Support of Large-Scale Science}, pages={12}, year={2009}, organization={ACM} }	98	Read
	Abstract: MapReduce provides a parallel and scalable programming model for data-intensive business and scientific applications. MapReduce and its de facto open source project, called Hadoop, support parallel processing on large datasets with capabilities including automatic data partitioning and distribution, load balancing, and fault tolerance management. Meanwhile, scientific workflow management systems, e.g., Kepler, Taverna, Triana, and Pegasus, have demonstrated their ability to help domain scientists solve scientific problems by synthesizing different data and computing resources. By integrating Hadoop with Kepler, we provide an easy-to-use architecture that facilitates users to compose and execute MapReduce applications in Kepler scientific workflows. Our implementation demonstrates that many characteristics of scientific workflow management systems, e.g., graphical user interface and component reuse and sharing, are very complementary to those of MapReduce. Using the presented Hadoop components in Kepler, scientists can easily utilize MapReduce in their domain-specific problems and connect them with other tasks in a workflow through the Kepler graphical user interface. We validate the feasibility of our approach via a word count use case.		

B.3 Search 3

Search Id	3
Search date	2014 December
Site	scholar.google.com
Search terms	data provenance mapreduce
Other	since 2013

Results

Id	Reference Description	Citations	Assessment
3.1	@incollection{glavic2014big, title={Big data provenance: challenges and implications for benchmarking}, author={Glavic, Boris}, booktitle={Specifying Big Data Benchmarks}, pages={72–80}, year={2014}, publisher={Springer} }	9	Read.
	Abstract: Data Provenance is information about the origin and creation process of data. Such information is useful for debugging data and transformations, auditing, evaluating the quality of and trust in data, modelling authenticity, and implementing access control for derived data. Provenance has been studied by the database, workflow, and distributed systems communities, but provenance for Big Data - which we refer to as Big Provenance - is a largely unexplored field. This paper reviews existing approaches for large-scale distributed provenance and discusses potential challenges for Big Data benchmarks that aim to incorporate provenance data/management. Furthermore, we will examine how Big Data benchmarking could benefit from different types of provenance information. We argue that provenance can be used for identifying and analyzing performance bottlenecks, to compute performance metrics, and to test a system's ability to exploit commonalities in data and processing.		
3.1.1	Reference 3.3		
3.1.2	Reference 2.10		
3.1.3	Reference 2.2		

3.2	@article{aji2013hadoop, title={Hadoop GIS: a high performance spatial data warehousing system over mapreduce}, author={Aji, Ablimit and Wang, Fusheng and Vo, Hoang and Lee, Rubao and Liu, Qiaoling and Zhang, Xiaodong and Saltz, Joel}, journal={Proceedings of the VLDB Endowment}, volume={6}, number={11}, pages={1009–1020}, year={2013}, publisher={VLDB Endowment} }	96	Not to read
	Abstract: Support of high performance queries on large volumes of spatial data becomes increasingly important in many application domains, including geospatial problems in numerous fields, location based services, and emerging scientific applications that are increasingly data- and compute-intensive. The emergence of massive scale spatial data is due to the proliferation of cost effective and ubiquitous positioning technologies, development of high resolution imaging technologies, and contribution from a large number of community users. There are two major challenges for managing and querying massive spatial data to support spatial queries: the explosion of spatial data, and the high computational complexity of spatial queries. In this paper, we present Hadoop-GIS – a scalable and high performance spatial data warehousing system for running large scale spatial queries on Hadoop. Hadoop-GIS supports multiple types of spatial queries on MapReduce through spatial partitioning, customizable spatial query engine RESQUE, implicit parallel spatial query execution on MapReduce, and effective methods for amending query results through handling boundary objects. Hadoop-GIS utilizes global partition indexing and customizable on demand local spatial indexing to achieve efficient query processing. Hadoop-GIS is integrated into Hive to support declarative spatial queries with an integrated architecture. Our experiments have demonstrated the high efficiency of Hadoop-GIS on query response and high scalability to run on commodity clusters. Our comparative experiments have showed that performance of Hadoop-GIS is on par with parallel SDBMS and outperforms SDBMS for compute-intensive queries. Hadoop-GIS is available as a set of library for processing spatial queries, and as an integrated software package in Hive.		
3.3 [31]	@inproceedings{akoush2013hadoopprov, title={HadoopProv: Towards Provenance as a First Class Citizen in MapReduce.}, author={Akoush, Sherif and Sohan, Ripduman and Hopper, Andy}, booktitle={TaPP}, year={2013} }	9	Read

	Abstract: We introduce HadoopProv, a modified version of Hadoop that implements provenance capture and analysis in MapReduce jobs. It is designed to minimise provenance capture overheads by (i) treating provenance tracking in Map and Reduce phases separately, and (ii) deferring construction of the provenance graph to the query stage. Provenance graphs are later joined on matching intermediate keys of the Map and Reduce provenance files. In our prototype implementation, HadoopProv has an overhead below 10% on typical job runtime (<7% and <30% average temporal increase on Map and Reduce tasks respectively). Additionally, we demonstrate that provenance queries are serviceable in O(k log n), where n is the number of records per Map task and k is the set of Map tasks in which the key appears.		
3.3.1	Reference 2.6		
3.3.2	Reference 2.11		
3.4	@inproceedings{sarma2013upper, title={Upper and lower bounds on the cost of a map-reduce computation}, author={Sarma, Anish Das and Afrati, Foto N and Salihoglu, Semih and Ullman, Jeffrey D}, booktitle={Proceedings of the VLDB Endowment}, volume={6}, number={4}, pages={277–288}, year={2013}, organization={VLDB Endowment} }	54	Not to read
	Abstract: In this paper we study the tradeoff between parallelism and communication cost in a map-reduce computation. For any problem that is not "embarrassingly parallel," the finer we partition the work of the reducers so that more parallelism can be extracted, the greater will be the total communication between mappers and reducers. We introduce a model of problems that can be solved in a single round of map-reduce computation. This model enables a generic recipe for discovering lower bounds on communication cost as a function of the maximum number of inputs that can be assigned to one reducer. We use the model to analyze the tradeoff for three problems: finding pairs of strings at Hamming distance d, finding triangles and other patterns in a larger graph, and matrix multiplication. For finding strings of Hamming distance 1, we have upper and lower bounds that match exactly. For triangles and many other graphs, we have upper and lower bounds that are the same to within a constant factor. For the problem of matrix multiplication, we have matching upper and lower bounds for one-round map-reduce algorithms. We are also able to explore two-round map-reduce algorithms for matrix multiplication and show that these never have more communication, for a given reducer size, than the best one-round algorithm, and often have significantly less.		

3.5	@article{chen2013big, title={Big data challenge: a data management perspective}, author={Chen, Jinchuan and Chen, Yueguo and Du, Xiaoyong and Li, Cuiping and Lu, Jiaheng and Zhao, Suyun and Zhou, Xuan}, journal={Frontiers of Computer Science}, volume={7}, number={2}, pages={157–164}, year={2013}, publisher={Springer} }	29	Not to read
	Abstract: There is a trend that, virtually everyone, ranging from big Web companies to traditional enterprisers to physical science researchers to social scientists, is either already experiencing or anticipating unprecedented growth in the amount of data available in their world, as well as new opportunities and great untapped value. This paper reviews big data challenges from a data management respective. In particular, we discuss big data diversity, big data reduction, big data integration and cleaning, big data indexing and query, and finally big data analysis and mining. Our survey gives a brief overview about big-data-oriented research and problems.		
3.6 [38]	@article{carata2014primer, title={A primer on provenance}, author={Carata, Lucian and Akoush, Sherif and Balakrishnan, Nikilesh and Bytheway, Thomas and Sohan, Ripduman and Selter, Margo and Hopper, Andy}, journal={Communications of the ACM}, volume={57}, number={5}, pages={52–60}, year={2014}, publisher={ACM} }	11	Read
	Abstract: Assessing the quality or validity of a piece of data is not usually done in isolation. You typically examine the context in which the data appears and try to determine its original sources or review the process through which it was created. This is not so straightforward when dealing with digital data, however: the result of a computation might have been derived from numerous sources and by applying complex successive transformations, possibly over long periods of time. As the quantity of data that contributes to a particular result increases, keeping track of how different sources and transformations are related to each other becomes more difficult. This constrains the ability to answer questions regarding a result's history, such as: What were the underlying assumptions on which the result is based? Under what conditions does it remain valid? What other results were derived from the same data sources?		

3.7	@article{talia2013toward, title={Toward Cloud-based Big-data Analytics}, author={Talia, Domenico}, journal={IEEE Computer Science}, pages={98–101}, year={2013} }	38	Not to read
	Abstract: Extracting useful knowledge from huge digital datasets requires smart and scalable analytics services, programming tools, and applications.		
3.8	@inproceedings{aji2013demonstration, title={Demonstration of hadoop-gis: A spatial data warehousing system over mapreduce}, author={Aji, Ablimit and Sun, Xiling and Vo, Hoang and Liu, Qioaling and Lee, Rubao and Zhang, Xiaodong and Saltz, Joel and Wang, Fusheng}, booktitle={Proceedings of the 21st ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems}, pages={528–531}, year={2013}, organization={ACM} }	9	Not to read
	Abstract: The proliferation of GPS-enabled devices, and the rapid improvement of scientific instruments have resulted in massive amounts of spatial data in the last decade. Support of high performance spatial queries on large volumes data has become increasingly important in numerous fields, which requires a scalable and efficient spatial data warehousing solution as existing approaches exhibit scalability limitations and efficiency bottlenecks for large scale spatial applications. In this demonstration, we present Hadoop-GIS – a scalable and high performance spatial query system over MapReduce. HadoopGIS provides an efficient spatial query engine to process spatial queries, data and space based partitioning, and query pipelines that parallelize queries implicitly on MapReduce. Hadoop-GIS also provides an expressive, SQL-like spatial query language for workload specification. We will demonstrate how spatial queries are expressed in spatially extended SQL queries, and submitted through a command line/web interface for execution. Parallel to our system demonstration, we explain the system architecture and details on how queries are translated to MapReduce operators, optimized, and executed on Hadoop. In addition, we will showcase how the system can be used to support two representative real world use cases: large scale pathology analytical imaging, and geo-spatial data warehousing.		

3.9	@inproceedings{che2013big, title={From big data to big data mining: challenges, issues, and opportunities}, author={Che, Dunren and Safran, Mejdl and Peng, Zhiyong}, booktitle={Database Systems for Advanced Applications}, pages={1–15}, year={2013}, organization={Springer} }	27	Not to read
	Abstract: While "big data" has become a highlighted buzzword since last year, "big data mining", i.e., mining from big data, has almost immediately followed up as an emerging, interrelated research area. This paper provides an overview of big data mining and discusses the related challenges and the new opportunities. The discussion includes a review of state-of-the-art frameworks and platforms for processing and managing big data as well as the efforts expected on big data mining. We address broad issues related to big data and/or big data mining, and point out opportunities and research topics as they shall duly flesh out. We hope our effort will help reshape the subject area of today's data mining technology toward solving tomorrow's bigger challenges emerging in accordance with big data.		
3.10	@article{mattmann2013computing, title={Computing: A vision for data science}, author={Mattmann, Chris A}, journal={Nature}, volume={493}, number={7433}, pages={473–475}, year={2013}, publisher={Nature Publishing Group} }	36	Not to read
	Abstract not available.		