

Ibis: A Provenance Manager for Multi-Layer Systems

Christopher Olston and Anish Das Sarma
Yahoo! Research

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; map-reduce 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.

1. INTRODUCTION

Modern systems are often comprised of multiple semi-independent sub-systems. Examples at Yahoo come in at least three varieties:

- **Stacked:** systems with higher-level abstractions stacked upon lower-level systems, e.g. Oozie [2] stacked on Pig [3] stacked on Hadoop [1].
- **Pipelined:** data flows through a sequence of systems, e.g. a system for ingesting RSS feeds, then a system for processing the feeds, then a system for indexing and serving the feeds via a search interface.
- **Side-by-side:** Two systems serving the same role might operate side-by-side during a migration period, with re-

sponsibility being transferred to a replacement system gradually, to allow the new system to be vetted and fine-tuned. In another scenario, redundant systems are deployed in a permanent side-by-side configuration, with each one targeting a different point in some performance tradeoff space such as latency vs. throughput.¹

Modularity in these forms facilitates the creation of complex systems, but can complicate operational issues, including monitoring and debugging of end-to-end data processing flows. To follow a single RSS feed from beginning to end may require interacting with half a dozen sub-systems, each of which likely has different metadata and different ways of querying it.

Yahoo architects would like to reduce the manual effort required to track data across sub-systems. Solutions that rely on standardization efforts or deep code modifications are undesirable, and in fact unrealistic when using third-party components, or even in-house ones that are already mature and widely deployed.

Motivated by this challenge, we are creating *Ibis*, a service that collects, integrates, and makes queryable the metadata produced by different sub-systems in a data processing environment. This approach has three main advantages:

- Users are provided with an integrated view of metadata, via a uniform query interface.
- Boilerplate code for storing and accessing metadata is factored out of n data processing sub-systems, into one place (*Ibis*). Moreover, since *Ibis* specializes in metadata management it will likely do a better job, versus the data processing sub-systems for which metadata management falls into the “bells and whistles” category.
- The lifespan of the metadata is decoupled from that of the data to which it refers, and even from the lifespans of the various data processing sub-systems.

1.1 Provenance Metadata Heterogeneity

Arguably the most complex type of metadata to manage is *data provenance*, which is the focus of this paper. A system that aims to integrate provenance metadata from multiple sub-systems must deal with nonuniformity and incompleteness.

¹For example, one might find a low-latency/low-throughput feed processing engine deployed side-by-side with a high-latency/high-throughput engine, with time-sensitive feeds (e.g. news) handled by the former and the majority of feeds handled by the latter.

To begin with, different sub-systems often represent data and processing elements at different granularities. Data granularities range from tables (coarse-grained) to individual cells of tables (fine-grained), with multiple possible mid-granularity options, e.g. rows vs. columns vs. temporal versions. Process descriptions also run the gamut from coarse-grained (e.g. a SQL query or Pig script) to fine-grained (e.g. one Pig operator in one retry attempt of one map task), also with multiple ways to sub-divide mid-granularity elements (e.g. map and reduce phases vs. Pig operations (which may span phases) vs. parallel partitions).

Moreover, links among processing and data elements sometimes span granularities. For example, one system at Yahoo records a link from each (row, column group, version) combination (e.g. latest release date and opening theater for the movie “Inception”) to an external source feed (e.g. Rotten Tomatoes).

Finally, one cannot assume that each sub-system gives a complete view of its metadata. At Yahoo, and presumably elsewhere, metadata recording is enhanced over time as new monitoring and debugging needs emerge. Recording “all” metadata at the finest possible granularity sometimes imposes unacceptable implementation and performance overheads on the system producing the metadata, as well as on the system capturing and storing it.

Ibis accommodates these forms of diversity and incompleteness with a *multi-granularity provenance model* coupled with query semantics based on an *open-world assumption* [9]. This paper describes Ibis’s provenance model, query language and semantics, and gives examples of their usage. The model and language have been fully implemented on top of an ordinary RDBMS, using simple query rewriting techniques. Performance and scalability issues are subjects of ongoing work, and are not the focus of the present paper.

1.2 Outline

The remainder of this paper is structured as follows. We discuss related work in Section 2. Then we present Ibis’s provenance data model in Section 3. The semantics and syntax of Ibis’s query language are given in Sections 4–6. We describe a prototype implementation of a storage and query manager for Ibis in Section 7.

2. RELATED WORK

Provenance metadata management has been studied extensively in the database [5] and scientific workflow [7] literature, including the notion of offering provenance management as a first-class service, distinct from data and process management, e.g. [14]. Many aspects of our approach borrow from prior provenance work, and are somewhat standard at this point. For example, modeling provenance relationships as (source data node, process node, target data node) triples, use of free-form key/value attributes, and use of a declarative SQL/datalog-style query language, are all commonalities between Ibis and other approaches such as Kepler’s COMAD provenance manager [4]. However, most prior work on provenance has focused on tracking a single system’s provenance metadata, and consequently has generally assumed that provenance metadata is rather uniform, and/or can be tightly coupled to the data in one system.

We provide the first formal framework—data provenance model, query language and semantics—for integrated management of provenance metadata that spans a rich, multi-

dimensional granularity hierarchy. The core contribution of our work is a set of rules for inferring provenance relationships across granularities. These inference rules have carefully-chosen, precise semantics and have been embedded in our query language and system.

Several prior projects offer (restricted) multi-granularity models, but none of them focus on formal semantics for inferring relationships when provenance is queried:

- Kepler’s COMAD model [4] and ZOOM user views [6] deal with uni-dimensional granularity hierarchies in data (COMAD’s nested collections) or process (ZOOM’s sub-workflows), but neither supports multi-dimensional granularity hierarchies, a combination of data and process hierarchies, or the ability for queries to infer provenance relationships across granularities.
- The *open provenance model* [10] shares our goal of offering a generic framework for representing and accessing provenance metadata from diverse sources. The open provenance model aims to support multi-dimensional granularity hierarchies via the notion of “refinement,” but it does not provide specific semantics for multi-granularity refinement, or formal rules for carrying provenance relationships across granularities in the data model, query language or system.
- References [8, 13] consider annotations on arbitrary two-dimensional sub-regions of relational tables, but do not deal with provenance linkage and inference.

One branch of the Harvard PASS project [11] shares our goal of managing provenance that spans system layers. That work restricts its attention to coarse-grained provenance, and tackles numerous issues around capturing and cleaning the provenance metadata (e.g. APIs, object naming schemes and cycle detection). It is complementary to the work we present in this paper, which considers multi-granularity provenance and focuses on how to represent and query it. As our project moves forward to tackle the capture and cleaning issues, we hope to leverage the PASS work.

Lastly, Ibis supports relatively simple forms of provenance—where-provenance and lineage (“flat” why-provenance)—which suffice for most use-cases we have encountered at Yahoo. More elaborate forms of provenance that associate logic expressions with provenance links (e.g. witness sets and how-provenance; see [5]) are not our focus.

3. DATA PROVENANCE MODEL

This section introduces Ibis’s model of provenance graphs that connect data and process elements at various granularities.

3.1 Data and Process Granularities

An Ibis instance is configured with *granularity sets* (gsets) that describe the possible granularities of data and process elements and their containment relationships.

DEFINITION 3.1 (GSET). *A gset is defined by a bounded partially-ordered set (poset) $G = (\mathbf{G}, \preceq, g_{\max}, g_{\min})$, where \mathbf{G} gives the finite set of granularities, “ \preceq ” denotes containment and defines a partial order over \mathbf{G} , and there exist unique maximal and minimal elements $g_{\max}, g_{\min} \in \mathbf{G}$, i.e., $\forall g \in \mathbf{G} : g_{\min} \preceq g \preceq g_{\max}$.*

Figure 1 gives example data and process gsets, which are based on some of Yahoo’s web data management applications. The arrows denote containment relationships: an

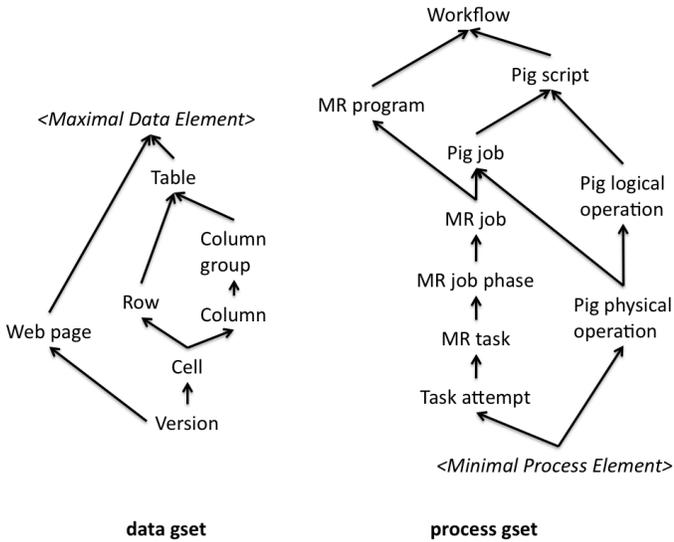


Figure 1: Example gsets.

arrow from X to Y denotes $X \preceq Y$. Intuitively, $X \preceq Y$ implies that each element at granularity Y must have an element at finer-granularity X , but the converse may not hold, i.e., each element at granularity X does not need to have an element at coarser-granularity Y .

In our example, data is either part of a relational table or a free-form web page. Relational tables are divided horizontally into rows, and vertically into column groups, which are further subdivided into columns. A row/column combination is a cell. A table cell can have multiple versions of data, e.g. reflecting multiple conflicting possible data values, or temporally changing values. Web pages also have versions, corresponding to multiple crawled snapshots.

Processing, at the coarsest granularity, is driven by workflows whose steps are either map-reduce programs or pig scripts. An execution of a program or script is called a job. Pig jobs are comprised of a series of map-reduce jobs, which are in turn broken into two phases (map and reduce). Each phase is partitioned into map or reduce tasks, which undergo one or more execution attempts. Syntactically, Pig scripts consist of sequences of logical operations. Pig logical operations are compiled into sequences of physical operations, which perform the work inside the map/reduce task attempts.²

When a new Ibis instance is configured, one data gset and one process gset must be supplied. If unique maximal and minimal elements are absent from either gset, Ibis creates them automatically (e.g. *<Maximal Data Element>* and *<Minimal Process Element>* in Figure 1).

3.2 Data and Process Elements

We start by defining *basic elements*, the atomic unit of a data or process item. Each basic element is specified by a particular granularity, a unique identifier, and parent (coarser-granularity) basic elements as defined below.

DEFINITION 3.2 (BASIC ELEMENT). A basic element $b = (g, id, \mathcal{P})$ is defined by a granularity g in the data or process

²In general there is no containment relationship between Pig operations and map/reduce tasks, or even map/reduce phases (e.g. some join operations span phases).

gset, a globally unique³ id , and a set \mathcal{P} of ids of basic elements that are direct parents in the containment hierarchy.

Our next definition formalizes the notion of containment of basic elements:

DEFINITION 3.3 (BASIC ELEMENT CONTAINMENT). Given two basic elements $b_1 = (g_1, id_1, \mathcal{P}_1)$ and $b_2 = (g_2, id_2, \mathcal{P}_2)$, b_1 contains b_2 iff either $id_1 \in \mathcal{P}_2$ or $\exists b^* \in \mathcal{P}_2$ such that b_1 contains b^* (according to recursive application of this definition).

Intuitively, b_2 is contained in b_1 if b_1 is a direct parent (i.e., coarser granularity element) or an ancestor in the granularity hierarchy.

Next we define the notion of “granularizing” basic elements to the finest possible granularity, a concept that will be used later to infer new relationships among elements. Granularization simply consists of finding all basic elements of the finest granularity contained in a given element:

DEFINITION 3.4 (BASIC ELEMENT GRANULARIZATION). Given basic element $b = (g, id, \mathcal{P})$ and minimal element g_{\min} in the gset containing g , the granularization of b , written $\mathcal{G}(b)$, is $\{b' = (g_{\min}, id', \mathcal{P}') : b \text{ contains } b'\}$.

Next we define the notions of complex element types and complex elements, which allow us to compose elements from multiple basic elements of different granularities.

DEFINITION 3.5 (COMPLEX ELEMENT TYPE). A complex element type $T = \{g_1, g_2, \dots, g_n\}$ is a set of granularities such that all members are from the same gset (i.e. all data granularities or all process granularities) and no two members $g_i, g_j \in T$ satisfy $g_i \preceq g_j$.

An example complex element type is $\{\text{row}, \text{column group}\}$, which denotes a data element defined by the intersection of a particular row and a particular column group. Each complex element type has an associated *attribute set* $A = \{a_1, a_2, \dots, a_m\}$, $m \geq 0$, e.g. $\{\text{owner}, \text{storage location}\}$.

DEFINITION 3.6 (COMPLEX ELEMENT). A complex element $E = (id, T = \{g_1, g_2, \dots, g_n\}, \{b_1, b_2, \dots, b_n\})$ consists of a globally unique id , a type T , and a basic element b_i corresponding to each granularity $g_i \in T$.

An example complex element is $(8, \{\text{row}, \text{column group}\}, \{\text{row 5}, \text{column group 3}\})$.

Finally, we extend the definition of granularization to complex elements, in the natural way:

DEFINITION 3.7 (COMPLEX ELEMENT GRANULARIZATION). Given complex element $E = (id, T, \{b_1, b_2, \dots, b_n\})$, the granularization of E is $\mathcal{G}(E) = \bigcap_{1 \leq i \leq n} \mathcal{G}(b_i)$.

3.3 Provenance Graph

Ibis manages a *provenance graph* that relates sets of complex elements to one another via three-way relationships. Figure 2 shows an example provenance graph from a simple web information extraction scenario in which movie data

³Our model can be extended easily to accommodate scoped ids, e.g. row ids that are unique within the scope of a table would be identified via (table id, row id) pairs.

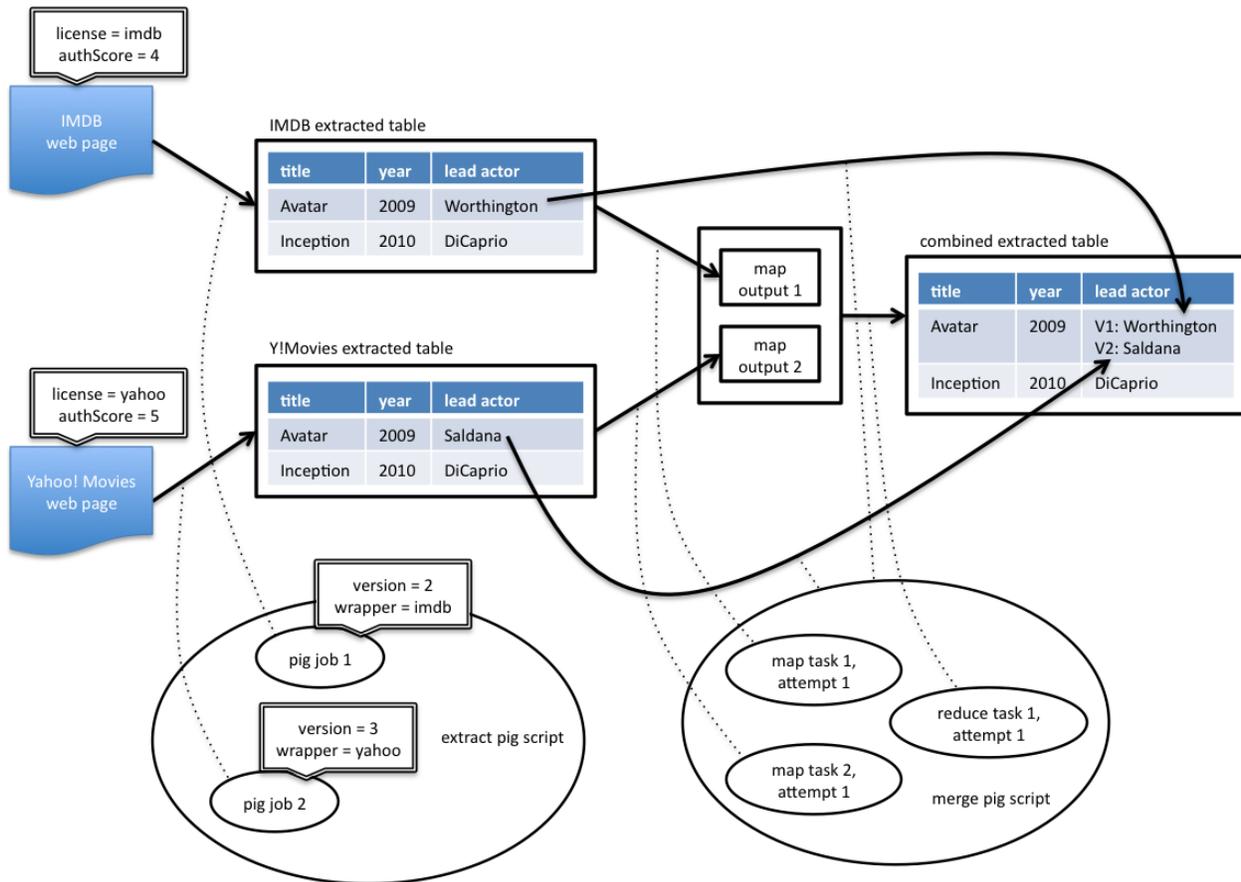


Figure 2: Example provenance graph.

has been extracted from two web pages (IMDB and Yahoo! Movies) and then merged. Inconsistencies have been preserved, and stored as alternate versions of cells in the merged table (the two web pages differed on the lead actor for the film “Avatar”).

Formally, a graph vertex $V = (id, T, \{e_1, e_2, \dots, e_k\}, \{v_1, v_2, \dots, v_m\})$ is defined by a globally unique id, a type T , the ids of one or more complex element e_i of type T , and a value v_j for each attribute in T 's attribute set. Each vertex represents the union of a set of complex data or process elements of a given type. A common case involves sets of size one ($k = 1$), e.g. $(12, \{\text{row, column group}\}, \{8\}, \{\text{owner "Jeff", location "Singapore data center"}\})$. Another example, with $k = 2$ (but no attribute values, also common), is $(14, \{\text{MR task}\}, \{9, 10\}, \{\})$ where 9 and 10 refer to complex elements $(9, \{\text{map task}\}, \{\text{map task 1}\})$ and $(10, \{\text{map task}\}, \{\text{map task 2}\})$, respectively. Figure 2 has one vertex with $k = 2$: the rectangle surrounding “map output 1” and “map output 2.” Most vertices in Figure 2 have no attributes; exceptions are: web pages (license and authority score); extract pig jobs (version of extract script used, wrapper parameter).

Connections among graph vertices take the form of three-way (d_1, p, d_2) relationships, denoting that process element p produced data element d_2 by reading data element d_1 . More particularly, *part of* p produced *all of* d_2 by reading *part of* d_1 . (These semantics stem from the fact that creation of

a data “touches” every byte of the data, whereas reading data and executing code rarely touch all the data/code (e.g. indexes and column stores; code branches).)⁴

In Figure 2, each provenance relationship (d_1, p, d_2) is shown as a dark arrow (d_1 to d_2 link) combined with a light dotted arrow (link to p). The provenance relationships on the left-hand side of Figure 2 are coarse-grained in terms of data links, and semi-coarse-grained in terms of process links (pig jobs that ran a particular version of the pig script called “extract,” with a particular web page wrapper). The provenance relationships on the right-hand side occur at two granularities: (1) fine-grained links from data cells in the IMDB and Yahoo! Movies tables to cell versions in the combined extracted table, with coarse-grained references to the “merge” pig script; (2) coarse-grained links from the IMDB and Yahoo! Movies tables to the combined extracted table (via intermediate map output files), with fine-grained references to the specific map and reduce task attempts that handled the data.

4. OPEN-WORLD SEMANTICS

Recall from Section 1.1 that Ibis makes an *open-world* assumption about the metadata it manages. Here we formally

⁴We have found these semantics to suffice for the applications we have considered, but of course if needed one could always expose the control over the *part/all* semantics of each provenance connection to users.

define open-world semantics in the context of Ibis.

Let M denote the metadata currently registered with an Ibis instance. M encodes a set \mathcal{F} of facts, such as the known set of data and process elements, their containment relationships, and the known provenance linkages. Ibis assumes that \mathcal{F} is *correct* but (possibly) not *complete*, i.e. there exists some *true world* of facts $\hat{\mathcal{F}} \supseteq \mathcal{F}$. Let the *extension* $ext(\mathcal{F})$ denote the set of all facts that can be derived from \mathcal{F} and are guaranteed to be part of any true world that is consistent with \mathcal{F} , i.e. $\mathcal{F} \subseteq ext(\mathcal{F}) \subseteq \hat{\mathcal{F}}$. ($ext(\mathcal{F})$ consists of all *certain* facts, analogous to *certain answers* in standard open-world semantics [9].)

Examples of facts in $ext(\mathcal{F})$ that are not in \mathcal{F} include inferred containment relationships for complex elements, and transitively inferred provenance links. As an example of a fact that may be in $\hat{\mathcal{F}}$ but is not in $ext(\mathcal{F})$, suppose \mathcal{F} includes “process p emitted row r_1 ,” “process p emitted row r_2 ,” and “ r_1 and r_2 are part of table T ”; even if \mathcal{F} mentions no rows in T other than r_1 and r_2 , the assertion “process p emitted the entire table T ” cannot be included in $ext(\mathcal{F})$ because of the possibility that T contains additional rows in the true world $\hat{\mathcal{F}}$.

Ibis queries are answered with respect to $ext(\mathcal{F})$. In other words, the answer to query Q is equivalent to the one produced by the following two-step evaluation procedure: (1) derive and materialize $ext(\mathcal{F})$; (2) answer Q by performing “lookups” into $ext(\mathcal{F})$. These steps are the subjects of Sections 5 and 6, respectively.

Note that “positive queries” (which lookup facts that are implied by $ext(\mathcal{F})$) yield certain-answers semantics, but “negative queries” (which lookup facts that are not implied by $ext(\mathcal{F})$) such as ones that use “NOT EXISTS” or “MAX” do not, because some facts that cannot be derived based on Ibis’s knowledge may be correct in the true state of the world. For completeness, our query language described in Section 6 does permit negative constructs. In practice they should either be disallowed, or come with a disclaimer about the deviation from certain-answers semantics. Another possibility is to record facts about completeness (i.e. the relationship between $ext(\mathcal{F})$ and $\hat{\mathcal{F}}$) such as “all table/job-level provenance links are being captured,” and use them to vet negative queries; developing such an approach is left as future work.

5. INFERRING RELATIONSHIPS

Ibis’s core strength is its ability to infer relationships among components of the provenance graph that span granularities. This section gives formal definitions of predicates that Ibis can infer with certainty (i.e. $ext(\mathcal{F})$, defined in Section 4). Let \mathcal{V} denote the set of provenance graph vertices currently registered with an Ibis instance. Under open-world semantics (Section 4) we must assume the existence of some vertex set $\mathcal{V}' \supseteq \mathcal{V}$ (along with additional provenance relationships) that captures the real situation. Ibis’s relationship inference semantics are defined in the context of \mathcal{V}' .

5.1 Under

Central to reasoning about granularity-spanning metadata is the *under* predicate, which determines whether the data or process element described by one vertex V_1 is contained in the element described by another vertex V_2 . For

example, in Figure 2 the cell containing **Worthington** is under the IMDB extracted table’s **lead actor** column, which in turn is under the IMDB extracted table.

DEFINITION 5.1 (UNDER). *Given two provenance graph vertices V_1 and V_2 with complex element sets $E(V_1)$ and $E(V_2)$, V_1 is under V_2 iff $\nexists \mathcal{V}' \supseteq \mathcal{V}$ such that $\bigcup_{e_1 \in E(V_1)} \mathcal{G}(e_1) \not\subseteq \bigcup_{e_2 \in E(V_2)} \mathcal{G}(e_2)$ ⁵.*

Fortunately, there exists an efficient way of checking whether a pair of vertices satisfies the under predicate using just the known vertex set \mathcal{V} , which is equivalent to the above definition (a proof of equivalence is given in Appendix A):

DEFINITION 5.2 (EFFICIENT UNDER CHECK). *Given two provenance graph vertices V_1 and V_2 with complex element sets $E(V_1)$ and $E(V_2)$, V_1 is under V_2 iff $\forall e_1 \in E(V_1), \exists e_2 \in E(V_2)$ such that e_1 is under e_2 , with the under predicate defined over complex elements as follows: Given two complex elements e_1 and e_2 with basic element sets $B(e_1)$ and $B(e_2)$, e_1 is under e_2 iff $\forall b_2 \in B(e_2), \exists b_1 \in B(e_1)$ such that b_2 contains⁶ b_1 .*

5.2 Feeds, Emits and Influences

Recall the three-way provenance relationships introduced in Section 3.3: a relationship (d_1, p, d_2) denotes that (part of) processing element p produced (all of) data element d_2 by reading (part of) data element d_1 . Ibis can answer three types of predicates over the set of registered provenance relationships:

- **Data feeding a process:** Given data element d and process element p , does (part of) d feed (part of) p ?
- **A process emitting data:** Given data element d and process element p , does (part of) p emit (all of) d ?
- **Data influencing other data:** Given two data elements d_1 and d_2 , does (part of) d_1 influence (all of) d_2 , either directly (*influences(1)*) or indirectly (*influences(k)*)?

Formal definitions and examples follow:

DEFINITION 5.3 (FEEDS). *Data vertex d feeds process vertex p iff there exists a provenance relationship (d', p', d_2) such that d' is under d and p' is under p .*

In our example provenance graph shown in Figure 2, from the relationship (IMDB web page, pig job 1, IMDB extracted table) we can infer that (part of) IMDB web page feeds (part of) extract pig script. From the relationship (Worthington, merge pig script, V1: Worthington) we can infer that (part of) row(Avatar, 2009, Worthington) feeds (part of) merge pig script.

DEFINITION 5.4 (EMITS). *Process vertex p emits data vertex d iff there exists a provenance relationship (d_1, p', d') such that p' is under p and d is under d' .*

Again considering Figure 2, from the relationship (IMDB web page, pig job 1, IMDB extracted table) we can infer that (part of) extract pig script emits (all of) IMDB extracted table, and also that (part of) pig job 1 emits (all of) row(Avatar, 2009, Worthington).

DEFINITION 5.5 (INFLUENCES). *Given two data vertices d_1 and d_2 : d_1 influences(0) d_2 iff d_2 is under d_1 ;*

⁵Recall the definition of granularization ($\mathcal{G}(\cdot)$) from Section 3.2.

⁶Recall the containment definition from Section 3.2.

d_1 influences(1) d_2 iff d_1 influences(0) d_2 or there exists a provenance relationship (d'_1, p, d'_2) such that d_1 influences(0) d'_1 and d'_2 influences(0) d_2 ; for any integer $k > 1$, d_1 influences(k) d_2 iff there exists a vertex d^* such that d_1 influences(1) d^* and d^* influences($k - 1$) d_2 .

The influence relationships in Figure 2 include:

- (part of) IMDB extracted table influences(0) (all of) row(Avatar, 2009, Worthington)
- (part of) row(Avatar, 2009, Worthington) influences(1) (all of) V1: Worthington
- (part of) IMDB web page influences(1) (all of) lead actor column of IMDB extracted table
- (part of) IMDB web page influences(2) (all of) V1: Worthington

An example of an inference that cannot be made is: (part of) IMDB extracted table influences(k) (all of) combined extracted table (for any value of k).

6. QUERY LANGUAGE

We now turn to Ibis’s query language, called *IQL*. Given the concepts introduced above and knowledge of SQL, the query language itself is fairly straightforward. Therefore, in lieu of a tedious exhaustive description of IQL, we give an overview of the main language constructs and illustrate their use via a number of examples.

IQL starts with SQL and makes the following modifications:

- The FROM clause references complex element types, e.g. Row or (Row, Column). IQL also supports special wildcards: AnyData (any data type), AnyProcess (any process type) and Any (any data or process type).
- The SELECT and WHERE clauses can reference a special id field, as well as elements of each type’s attribute set (for wildcards, no attributes are accessible).
- The *union*, *feeds*, *emits* and *influences(k)*⁷ predicates (defined in Section 5) can be used in the WHERE clause.

Table 1 gives some example query/answer pairs formulated over the example provenance graph in Figure 2. The first four queries in the table are inspired by data workflow debugging scenarios encountered at Yahoo. The fifth query corresponds to a Yahoo use-case involving content licensing: each data source comes with a license that restricts the contexts in which data derived from it can be displayed to end-users, and provenance is used to perform last-mile filtering for a given context. The final query is a somewhat contrived variation of the license example, which instead filters by source authority score; it shows a more elaborate use of our language.

Logically speaking, IQL queries are evaluated over the extended database $ext(\mathcal{F})$, defined in Section 4. As with SQL, IQL query semantics are equivalent to the following three-step evaluation strategy (in IQL’s case, over $ext(\mathcal{F})$): (i) evaluate the FROM clause to construct the cross-product of the sets of elements referenced; (ii) apply the filters given in the WHERE clause; and (iii) apply the projections specified in the SELECT clause.

⁷We do not currently support unbounded transitive closure ($k = \infty$), but support for this feature could be added via recursive query processing techniques.

7. PROTOTYPE IMPLEMENTATION

We have built a simple implementation of a storage and query manager for Ibis, on top of a conventional relational database system (SQLite [12]) using query rewriting from IQL into SQL. The purpose of this implementation is to test the applicability and ease-of-use of our model and query language, not to serve as an efficient or scalable system for managing provenance metadata—that is future work.

7.1 Relational Encoding

Our prototype uses a very simple encoding of Ibis’s information into relational tables:

- **Gsets:** The gsets are stored using two tables: `gnodes(type, granularity)` stores granularity nodes in the attribute `granularity`, with `type` being ‘Data’ or ‘Process’; each edge depicting a containment relationship between nodes of gsets are stored in table `gcont(child, parent)` in the obvious way.
- **Simple Elements:** The `simpleElements(seId, granularity)` table stores for every simple element identified by `seId`, the granularity in the gset given by `granularity`.
- **Complex Elements:** Table `complexElements(ceId, seId)` stores the mapping from complex elements to simple elements: a complex element `ceId` comprised of n simple elements is represented as n tuples $(ceId, seId_i)$.
- **Provenance Graph:** Vertices of the provenance graph (corresponding to sets of complex elements) are stored in table `vertices(vertexId, ceId)`, which associates a set of complex ids with each vertex, analogous to the way `complexElements` associates a set of simple ids with each complex id. Three-way provenance relationships are represented in the table `edges(src_data, process, dst_data)`, where `src_data` and `dst_data` are the source and destination data vertex identifiers and `process` gives the process vertex identifier.
- **Attributes:** Every complex element type `X` has a separate table `X.attrs(nodeid, <attributes>)` to store attributes. For example, in our demo scenario we have a table `Webpage.attrs(vertexId, license, authScore)` that maintains the attributes `license` and `authScore` for every webpage identified by `vertexId`.
- **Under:** Under relationships for simple elements are maintained using the `under(src, dst)` table in the obvious way.

7.2 Query rewriting

The four Ibis-specific constructs used to augment SQL—`under`, `influences`, `feeds`, `emits`—are automatically converted to SQL as follows. The `under` construct for vertices in the provenance graph (representing sets of complex elements) is converted to a SQL query over the `under` table (over simple elements) using the check from Definition 5.2. `Feeds` and `emits` are easily converted based on a lookup of the provenance tables. Note, however, that the translation of `feeds` and `emits` also needs to add the `under` table: For example, recall from Definition 5.3 that data vertex d feeds process vertex p when there is a provenance relationship (d', p', d_2) such that d' is under d and p' is under p . Finally, `influences(k)` is translated to a chain of joins using `under` and `edges`.

description	query	answer(s)
Find web pages that influence V1 of the Avatar lead actor field in the combined extracted table.	<code>select p.id from WebPage p, Version v where p influences(2) v and v.id = (Avatar lead actor V1);</code>	IMDB web page
Find data items that influence the combined extracted table.	<code>select d.id from AnyData d, Table t where d influences(2) t and t.id = (combined extracted table);</code>	{ map output file 1, map output file 2 } combined extracted table
Suppose the first attempt of the second map task of the merge job experienced a problem that is suspected to stem from malformed input data. Find the data table it read.	<code>select t.id from MRTaskAttempt a, Table t where t feeds a and a.id = (merge pig script map task 2 attempt 1);</code>	Y!Movies extracted table
Suppose version 3 of the extract pig script is found to have a bug. Find all “contaminated” data tables, i.e. ones containing data that stems from that version of the script.	<code>select t.id from PigScript p, PigJob j, AnyData d1, AnyData d2, Table t where p.id = (extract pig script) and j under p and j.version = 3 and j emits d1 and d1 influences(2) d2 and d2 under t;</code>	Y!Movies extracted table combined extracted table
Filter versions of combined table cells—only retain ones derived solely from sources with the Yahoo license.	<code>select v.id from Version v, Table t where t.id = (combined extracted table) and v under t and not exists (select * from WebPage source where source influences(2) v and source.license != ‘yahoo’);</code>	Avatar lead actor V2
Resolve cell version ambiguity by selecting the one that derives from the most authoritative web page.	<code>select v.id, source.authScore from Version v, WebPage source where source influences(2) v and source.authScore = (select max(source2.authScore) from Version v2, WebPage source2, (Row,Column) commonParent where source2 influences(2) v2 and v under commonParent and v2 under commonParent);</code>	id = Avatar lead actor V2; authScore = 5

Table 1: Example IQL queries, and the answers with respect to Figure 2.

8. CONCLUSION

Motivated by data processing systems comprised of multiple independently-developed sub-systems, we have developed a metadata and data provenance management service called Ibis. Ibis handles provenance metadata that spans multiple granularities of data and processing elements, and Ibis’s query language is able to infer provenance relationships across granularities. Ibis is fully implemented using query rewriting on top of a conventional RDBMS. Future work will focus on efficiency and scalability issues—in regard to storing and querying provenance metadata, as well as capturing and shipping it from various systems.

Acknowledgments

We would like to thank the various members of the Yahoo content platform teams who shared information about debugging, license management and data provenance. We also thank Philip Bohannon, Andreas Neumann and Kenneth Yocum for their input on this project.

9. REFERENCES

- [1] Apache. Hadoop: Open-source implementation of MapReduce. <http://hadoop.apache.org>.
- [2] Apache. Oozie: Hadoop workflow system. <http://issues.apache.org/jira/browse/HADOOP-5303>.
- [3] Apache. Pig: High-level dataflow system for hadoop. <http://hadoop.apache.org/pig>.
- [4] S. Bowers, T. M. McPhillips, and B. Ludascher. Provenance in collection-oriented scientific workflows. *Concurrency and Computation: Practice and Experience*, 20(5):519–529, 2008.
- [5] J. Cheney, L. Chiticariu, and W.-C. Tan. Provenance in databases: Why, how, and where. *Foundations and Trends in Databases*, 1(4):379–474, 2009.
- [6] S. Cohen-Boulakia, O. Biton, S. Cohen, and S. Davidson. Addressing the provenance challenge using ZOOM. *Concurrency and Computation: Practice and Experience*, 20(5):497–506, 2008.
- [7] S. B. Davidson, S. C. Boulakia, A. Eyal, B. Ludascher, T. M. McPhillips, S. Bowers, M. K. Anand, and

- J. Freire. Provenance in scientific workflow systems. *IEEE Data Engineering Bulletin*, 30(4):44–50, 2007.
- [8] M. Y. Eltabakh, W. G. Aref, A. K. Elmagarmid, M. Ouzzani, and Y. N. Silva. Supporting annotations on relations. In *Proc. EDBT*, 2009.
- [9] A. Y. Levy. Answering queries using views: A survey. *VLDB Journal*, 10(4):270–294, 2001.
- [10] Moreau (Editor), L., Plale, B., Miles, S., Goble, C., Missier, P., Barga, R., Simmhan, Y., Futrelle, J., McGrath, R., Myers, J., Paulson, P., Bowers, S., Ludaescher, B., Kwasnikowska, N., Van den Bussche, J., Ellkvist, T., Freire, J. and Groth, P. The Open Provenance Model (v1.01). *Technical Report*, 2008.
- [11] K.-K. Muniswamy-Reddy, U. Braun, D. A. Holland, P. Macko, D. Maclean, D. Margo, M. Seltzer, and R. Smogor. Layering in provenance systems. In *Proc. USENIX Annual Technical Conference*, 2009.
- [12] Open-Source Community. SQLite: An open-source database management library. <http://www.sqlite.org/>.
- [13] D. Srivastava and Y. Velegrakis. Intensional associations between data and metadata. In *Proc. ACM SIGMOD*, 2007.
- [14] M. Szomszor and L. Moreau. Recording and reasoning over data provenance in web and grid services. In *Proc. Intl. Conference on Ontologies, Databases and Applications of Semantics*, 2003.

APPENDIX

A. PROOF OF EQUIVALENCE OF UNDER DEFINITIONS

Here we prove the equivalence of Definition 5.1 and Definition 5.2. We start by proving the equivalence for the case when $E(V_1)$ and $E(V_2)$ each consists of a single complex element (Section A.1), then extend the proof to sets of complex elements (Section A.2).

A.1 Under for single complex elements

Let $E(V_1)$ and $E(V_2)$ contain single complex elements e_1 and e_2 respectively. From Definition 3.5, we know that there exists some $\mathcal{V}' \supseteq \mathcal{V}$ such that $\mathcal{G}(e_1) \neq \emptyset$. So we prove the result when $\mathcal{G}(e_1) \neq \emptyset$. From Definition 5.2, for the basic element sets $B(e_1)$ and $B(e_2)$, e_1 is under e_2 iff the following condition is satisfied: $\forall b_2 \in B(e_2), \exists b_1 \in B(e_1)$ such that b_2 contains b_1 . We prove the sufficiency and necessity of checking this condition:

Sufficient: For each $b_j \in B(e_2)$, let $b_{i(j)} \in B(e_1)$ satisfy the condition: That is, b_j is contained in $b_{i(j)}$. Therefore, we have

$$\mathcal{G}(b_{i(j)}) \subseteq \mathcal{G}(b_j)$$

Therefore, we have:

$$\mathcal{G}(e_1) = \bigcap_{b_i \in B(e_1)} \mathcal{G}(b_i) \subseteq \bigcap_{b_{i(j)} \in B(e_1)} \mathcal{G}(b_{i(j)}) \subseteq \bigcap_{b_j \in B(e_2)} \mathcal{G}(b_j) = \mathcal{G}(e_2)$$

Necessary: We prove necessity by contradiction. Suppose $b_j \in B(e_2)$ such that $\forall b_i \in B(e_1)$ we have that b_i is not contained in b_j , i.e., $\mathcal{G}(b_i) \not\subseteq \mathcal{G}(b_j)$. We have two cases: (1)

Suppose b_j is a basic element of the finest granularity, then e_1 cannot be under e_2 since $\mathcal{G}(e_1) \neq \emptyset$, $b_j \notin \mathcal{G}(e_1)$, and $\mathcal{G}(e_2) \subseteq \{b_j\}$. (2) Suppose b_j is a basic element of granularity coarser than g_{min} . Then consider the completion \mathcal{V}' of data obtained by adding distinct basic elements of granularity g_{min} under all basic elements of coarser granularity. We shall then have $\mathcal{G}(e_1) \not\subseteq \mathcal{G}(e_2)$, since $\mathcal{G}(b_j)$ will not contain $\mathcal{G}(e_1)$.

A.2 Under for sets of complex elements

Next we show that the equivalence of Definitions 5.1 and Definition 5.2 for sets of complex elements $E(V_1)$ and $E(V_2)$ easily follows from the proof of the single-element case. Below we show equivalence by proving the “if” and “only if” portions of the condition in Definition 5.2 separately.

If: Clearly if $\forall e_1 \in E(V_1), \exists e_2 \in E(V_2)$ such that e_1 is under e_2 (based on the condition for single elements), we have $\bigcup_{e_1 \in E(V_1)} \mathcal{G}(e_1) \subseteq \bigcup_{e_2 \in E(V_2)} \mathcal{G}(e_2)$.

Only If: We prove by contradiction. Suppose $\exists e_1 \in E(V_1)$ such that $\forall e_2 \in E(V_2)$ we have e_1 is not under e_2 , i.e., $\mathcal{G}(e_1) \not\subseteq \mathcal{G}(e_2)$. We can therefore construct \mathcal{V}' as follows. We add a simple element X of the finest granularity under e_1 such that $\forall e_2 \in E(V_2)$, we have $X \notin \mathcal{G}(e_2)$. Therefore, $X \in \bigcup_{e_1 \in E(V_1)} \mathcal{G}(e_1)$ but $X \notin \bigcup_{e_2 \in E(V_2)} \mathcal{G}(e_2)$. Therefore, $\bigcup_{e_1 \in E(V_1)} \mathcal{G}(e_1) \not\subseteq \bigcup_{e_2 \in E(V_2)} \mathcal{G}(e_2)$.