2009 International Conference on Network-Based Information Systems

Astro-WISE: Tracing and Using Lineage for Scientific Data Processing

Johnson Mwebaze, Danny Boxhoorn and Edwin Valentijn

Kapteyn Astronomical Institute, University of Groningen, Landleven 12, 9700 AV Groningen, The Netherlands
Telephone: + 31 50 363{4060, 4056, 4011}, Fax: +31 50 3636100
Email: {jmwebaze, d.r.boxhoorn, valentyn}@astro.rug.nl

***Abstract*—Most workflow systems that support data provenance primarily focus on tracing lineage of data. Data provenance by data lineage provides the derivation history of data including information about services and input data that contributed to the creation of a data product. We show that tracing lineage by means of full backward chaining not only enables users to share, discover and reuse the data, but also supports scientific data processing through storage, retrieval and (re)processing of digitized scientific data. In this paper, we present Astro-WISE, a distributed system for processing, analyzing and disseminating wide field imaging astronomical data. We show how Astro-WISE traces lineage of data and how it facilitates data processing, retrieval, storage, archiving. Particularly we show how it solves issues related to the changing data items typical for the scientific environment, such as physical changes in calibrations, our insight in these changes and improved methods for deriving results.**

***Keywords*-data lineage, persistence, provenance, target processing, dependencies**

I. Introduction

The nature of today's scientific experiments requires innovative dynamic approaches, in which results can be disseminated, re-derived, customized to each user's specific needs and shared between research groups. Scientific research involves distributed communities, running complex analysis techniques on data stored in heterogeneous archives. Such experiments are therefore characterized by repeatability of data derivation, support for collaboration, data sharing/reuse and the ability to verify data quality. Scientists always desire to verify the correctness of data or to follow through the derivation process to find out the cause of an abnormality that could have been discovered in the data. Repeating part of the derivation process could verify data quality, reveal the cause of the abnormality and also show which other datasets could have been affected by the same procedure. Thereafter, any new data generated as a result of such verification procedures does not replace old data but is stored as new data. Such procedures coupled with technological advances in the data acquisition techniques/equipment (e.g. design of sensor equipment like CCD Mosaics, wireless antennas) are creating accumulations of petabytes of raw and processed data. Then linking and maintaining dependencies between data (i.e. data lineage) becomes very important for the processing storage, archiving and retrieving of data. Following lineage data in the system, file storage, retrieval and archiving should be transparent to the application and or user.

Besides tracing lineage, e-Science systems should be flexible enough to enable researchers to work with and change methods on the fly while capturing all processing activities/events. Analogous to commercial databases that operate fixed programmes (e.g. mortgage plan) and variable data (e.g. interest), in a scientific environment the reverse applies. The processing methods and observational data continuously change. Scientists often focus on newly retrieved datasets for a certain period of time and also study variations between observational data. What happens when new releases of code or new computational methods become available? or there is new data for the same physical phenomena? Scientists may want to evaluate specific questions about the datasets, apply their own mining algorithms and visualization tools on datasets and derive their results following their insights. Preferably this knowledge is fed back into the system at the disposal of other researchers who optionally want to take advantage of the progress of this insight. It is therefore necessary to develop very high level abstractions of the data and the connection between dependent data items and use the connections to guide the researcher to knowledge, data and process.

The problem of tracing lineage (or provenance) of data has been extensively studied in the context of providing derivation history of data [1] [2]. We could leverage on existing lineage tracing techniques but our work goes beyond just tracing lineage of data and uses lineage data for scientific processing. Hence the current lineage tracing techniques fail short in two ways.

Firstly, provenance models proposed include workflow representation [3] [4], a description of data derivation and documentation of causal flow of events [5] [6]. Lineage data collected in such models is at the message level and therefore is insufficient as-is for re-using within the system to support scientific data processing. Lineage tracing in databases also assumes data is produced by relational operators and data stored in the database [7]. Accordingly, such methods fail to trace lineage when arbitrary programs are used and reside outside the database, which is typical of scientific applications. Other forms of lineage tracing include earlier

978-0-7695-3767-2/09 $25.00 © 2009 IEEE
DOI 10.1109/NBiS.2009.48

work in [8] which uses a weak reverse and verification function to compute a set of input data from a set of the output data. However today's scientific computations are complex therefore reversibility is not generally possible.

Secondly, most scientific applications use a Database Management System (DBMS) for data storage and the processing is done independent of the DBMS. Provenance models for such applications have been designed to capture the derivation process of a data item in the context of the application. Such applications fail to capture lineage for database manipulations done outside the application [9]. Accurate lineage is a combination of the scientific processes and database manipulations.

With distributed communities working on enormous data floods, scientific research demands for new approaches for managing and supporting scientific data processing. To address the above challenges, in our approach we load data into a database and make the database an integral part of all processing. The integration of the processing and the database ensures that all objects and references (links between all dependant data items) are stored and effectively become part of the system. By means of full backward chaining we leverage on lineage data for processing, retrieving and archiving of data. This paper presents the Astro-WISE Framework for data processing and data lineage. We show how Astro-WISE traces the lineage of data and how data lineage is used to facilitate data processing, retrieving, storage and archiving of data. Finally, we implemented the system and tested it with real astronomical applications.

The remainder of this paper is organized as follows. In Section II, we present an overview of Astro-WISE and give the details of Astro-WISE's framework for tracing lineage. We then show how data lineage is used to facilitate data processing in Section III. In Section IV, we show the implementation of the lineage framework. We test our implementation using a real astronomical application in Section V. Related work and the Conclusion appears in Section VI.

II. Astro-WISE

Astro-WISE enables astronomers to perform scientific experiments in a distributed environment. We make full use of the peer-to-peer architecture to explore and process data in the federated interconnected databases. The main components are: distributed storage, distributed databases and high-performance computing clusters. The dataservers (storage) and the databases store pixel data and metadata respectively. *Metadata includes any data other than pixel data*. The computing cluster decomposes and executes a job on a piece of hardware where the processing is optimal.

A. The Framework

Astro-WISE has been built to handle very large datasets and terabytes of catalogue data. It begins as a basic system

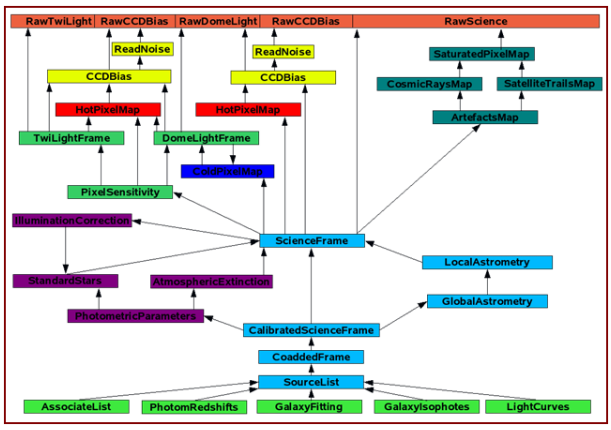


Figure 1. **A simplification of a target diagram: It allows processing of a target (an end data product) and any of its dependencies. The dependency chain is followed back towards the raw data (backward chaining, arrows as shown above) to make sure only those objects requiring processing are actually processed**

to an intelligent system ('quick look system') as new data and information is continuously processed and added to the system.

The core of the system exploits three properties in a database environment. Firstly, we apply the principle of inheritance using Object Oriented Programming (Python), where all Astro-WISE objects inherit key properties for database access, such as persistency of attributes. Secondly, the linking (associations or references) between instances of objects in the database is completely maintained. Thirdly, continuous growth of the database through the addition of new information or improvements made to existing information. We expand on these features in the remainder of this section.

B. Database Architecture and Persistence Objects

The Persistent Object Hierarchy is the core of Astro-WISE pipeline processing and data lineage. Processing is done through invocation of methods on these objects. Since these objects are persistent, all operations and attributes of these objects are automatically saved into a database.

All the I/O of the pipeline processes makes use of a database. This includes all information about the processing of data and the processing history. The database stores all persistent objects, attributes and raw data either as fully integrated objects or as descriptors. Only pixel values are stored outside the database in image and other data files. However, their unique filenames are stored in the database. Data can therefore only be manipulated through interaction with the database. A query to the database will provide all information related to the processing history and to the locations of all stored associated files, attributes and objects. Thus, the system provides the user with transparent access to all stages of the data processing and thereby allows the

data to be re-processed and knowledge fed back into the system. We utilize inbuilt security and authorization features of the database to enable sharing data in the system without breaking the dependency chain.

Access to the database is provided through Python. Python is used for both the Data Definition Language and the Data Manipulation Language. The database interface maps class definitions that are marked as persistent in Python to the corresponding SQL data definition statements.

C. Extendable Schema

The database provides an extendable schema that allows extending and modifying object attributes and method definitions through inheritance and polymorphism. This then allows end-users to define new persistent data products. Because schema modification complicates history tracking, we support schema evolution through inheritance and polymorphism to introduce new classes (or processing routines) to the pipeline. That way, any new processing routines can be defined and added. Users can modify functionality in modules, insert them into the system, or add a module on top of what currently exists, as long as these modules obey the standard data model.

D. Version Control and Preservation Management

The python source code is stored in a Concurrent Versioning System (CVS) database. Although CVS enables tracking versions of source code, CVS cannot distinguish between a method (subroutine) and a class in source code. It does not know that a couple of lines together form a method and can not tell if the changes made to the class/method substantially affects the meaning and structure of objects created through this class/method. At the lowest level, it is the connection between these classes and the created objects that we want to keep track of. We assign versions to classes through the class version attribute and use class descriptors to track version changes. All this is stored with the object (object versioning).

Preservation management addresses object inconsistencies (mismatches) created as a result of schema modification. An object mismatch occurs when the retrieving system tries to retrieve a particular object whose own generating class was different in the storing system, and whose structure is inconsistent with what the retrieving system is expecting. In a dynamic research environment, classes often change. Research prototypes propose a one-time conversion of old objects. Such a solution does not scale for a large persistent store. Astro-WISE has been designed to compare versions of data and classes. If there is an improved version of an object or a class, all dependent objects become outdated. A request for an outdated object will cause the object to be computed on-the-fly.

III. Data Lineage and Data processing

Astro-WISE follows a '*pull-based*' i.e. backward-chaining approach while processing data. This allows the end user to trace the data product, following all its dependencies up to the raw observational data and, if necessary, to re-derive the result with better data, and/or improved methods. It is unlike '*push-based*' i.e. forward-chaining systems, where the end user has little or no influence on what happens upstream. This paradigm is characterized by a fixed set of homogeneous, well-documented data products. Operators have a task to push the input data through the stream, often by means of a pipeline, irrespective of whether the derived data items are actually used by the end users.

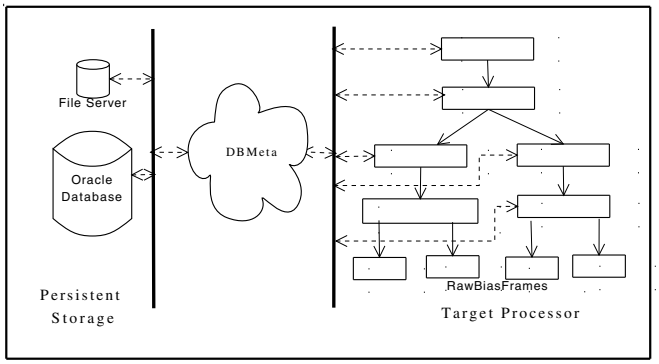


Figure 2. **The Figure shows the interaction between the *Target processor* and *DBObjectMeta* class which controls persistent class creation and delegates persistent object instantiation to the database. The Solid lines represent processing chain and the dashed arrows shows data movement to and from persistent storage and the boxes represent the different services(targets)**

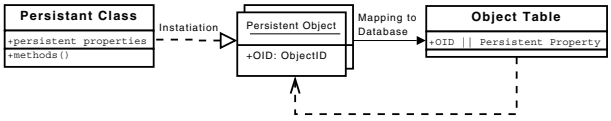


Figure 3. **Transparent mapping from a class definition to a table in a relational database. Attributes may refer to other persistent objects**

The Target Processor (TP)

Astro-WISE is built to handle queries by the user for his/her desired result, which we call a *Target*. The *Target* processor employs the dependency logic that is constructed using lineage data. Figure 1 shows a simplification of a Target diagram based on the system's object model.

The Target processor has three principal components: *service*, *pipeline* and a *Target*. A *service* is analogous to a class/method. *Pipelines* specify processing in terms of targets and dependencies. A *Target* is built by calling the appropriate service in the pipeline using a *make method* for that object. The *make method* follows the unix software development's *make* metaphor and describes the dependencies between objects. A *Target* is a database object representing a file or metadata that is passed as input to and generated as output from a process. For example, a Target could be

any calibrated image, or the results for a set of calibration parameters, or a list of parameter values describing an astronomical object.

Targets have dependencies that may themselves be targets. Targets are rebuilt in a recursive cascade as shown in Figure 4. If the requested target already exists, the system will check all its dependencies up to the raw data taken at the telescope. If all its dependencies are up-to-date the target object is returned. If all or some dependencies are not up-to-date, they will be recomputed on-the-fly or if new versions exist of the classes that created the objects, the target will also be recomputed.

Based on the information (data) in the database, the Target processor will reduce the processing tasks to only those required to generate new data products. In some cases the target processor will return the final data product if it already exists (i.e. processed before) and is up-to-date without any processing. A similar feature exists in Pegasus [10] called workflow reduction. However, Pegasus depends on the Replica Location Service (RLS) to determine which intermediate data has been registered and uses this data for processing. This method does not check the state of dependant data and therefore we cannot be sure that the final product is up-to-date.

IV. Tracing Lineage in Astro-WISE

A. Implementation

For each persistent class, a number of methods are defined which interact with the federated database. All class instantiations are automatically made persistent in the database forming an archive of all targets. To achieve this, the following major classes are implemented.

- `DBObject` is the root class of the hierarchy of the persistent classes. This class defines the primary key `object_id` of all objects.
- `DBObjectMeta` is the metaclass of `DBObject`. This is the class that is responsible for class creation and object instantiation.
- `DBProperties` is the module that defines all persistent attribute types which are defined by `persistent`.
- `DBSelect` implements a query language that is a natural extension to Python and that incorporates data lineage in the query syntax.
- `DBProxy` is an abstract interface to database vendor specific operations.

To use the persistency and query mechanisms only `DBObject` and `persistent` are required.

B. Lineage capture and Storage

We will show how data lineage is captured as dependencies between persistent objects that refer to another persistent object. The simplest persistent class has only one persistent attribute, like

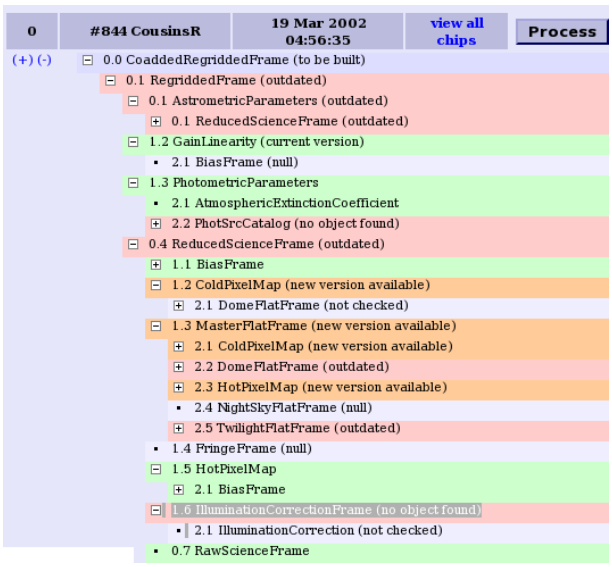


Figure 4. **Target processing: A tree view is given of the target(s). This tree view gives an overview of the target dependencies. Green dependencies are up-to-date, red dependencies are out-of-date and for orange dependencies indicate a newer version exists.**

```
class ClassA(DBObject):
  attribute = persistent('Description',
              attribute_type, default_value)
```

In this example `attribute` will have the type `attribute_type`. Atomic attribute types, such as `integer`, `string` or `float`, are translated into their database equivalents when an object is made persistent. If `attribute_type` is a subclass of a DBObject, the attribute will be a link to an object of that subclass. If `default_value` is an empty list, then the attribute will be an array of objects of `attribute_type`.

```
class ClassB(Object):
  e = persistent('A link', ClassA, None)
  f = persistent('List of links', ClassA, [])
  g = persistent('Link to object of ClassB')
```

`ClassB` defines three persistent attributes:

- `e` is a link to an instance of `ClassA`,
- `f` is a array of links to instances of `ClassA` (default empty), and
- `g` is a link to another instance of `ClassB`.

Invoking a procedure to make an object instantiates all classes recursively. This makes all objects persistent in the database forming a dynamic archive of all targets. The database thus has full awareness of all dependencies. See Figure 3 for the transparent mapping from the persistent class to a database table. Any definition, instantiation and querying of a persistent class is translated to the corresponding SQL statement. To support the storage of files in a similar way, the `DataObject` class is used.

```
class DataObject(DBObject):
  filename = persistent('File part', str, '')
```

The `filename` attribute is used to implement `store()` and `retrieve()` methods to transfer files to and from dataservers. Because the filename is kept in the database the storage and retrieval of files is transparent to the application. In the following example the file `example.txt` is made persistent by storing the file and committing the object of which it is part. Then we can search for all the files with a name that starts with `example` and fetch the first one (g[0])found from the dataservers.

```
example = DataObject(pathname='example.txt')
example.store()
example.commit()
g = DataObject.filename.like('example*')
g[0].retrieve()
```

C. Accessing and Querying Lineage Information

Given the system architecture and the implementation above, we can track any bit that was involved in the derivation of a data item. The end-to-end dependency linkages built into the system are used for query processing and data processing. Queries such as "*find the process that led to or that caused a data item to be as it is*" or "*Find out the 'on-the-fly' or processable dependencies which can be used to recreate a SourceList (sList)*" can be answered. Lineage data can be retrieved in three ways: (i) using Python scripts which are translated into SQL statements automatically, (ii) by using a web service object viewer tool and (iii) and using the dependency Cutout web service. Items (ii) and (iii) are demostrated in SectionV.

V. Experience with Astro-WISE

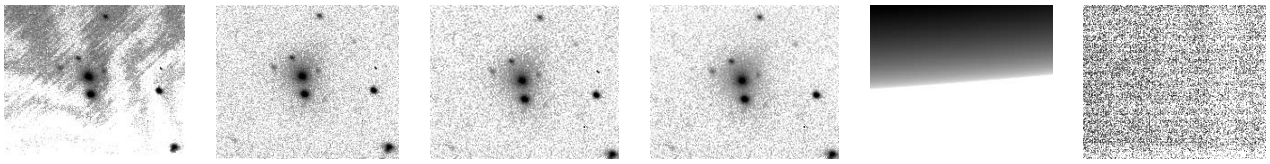
We used a case of *Creating and analyzing multi-dimensional data in Astro-WISE*. We used data observed by the ESO WFI@2.2meter telescope at La Silla, Chile. Data was observed for 12 nights spread over a period of 4 months. About 9 images of the field were taken for each night. This observational data was run through image reduction and calibration pipelines and co-added to produce a CoaddedRegriddedFrame (re-gridded composite of several science images) for each night. It is on this frame the science can be performed. We extract a list of sources (a SourceList) from this frame. Using an association tool, a cross-match between different SourceLists is made based on the position on the sky. This tool works fully in the database and creates database references lists called AssociateLists, which contain cross-references to matched sources. One AssociateList is made from two or more SourceLists. Quality control can be checked by comparing data from SourceLists taken on different nights. If we are happy with the quality, we associate all SourceLists to that of the first night to create one big AssociateList. Note that all lineage data is ingested into the database and everything here can be reprocessed. For purposes of this paper, the visualization steps have been omitted.

The processing described above is done recursively following dependencies to the raw observational data. We can (re)create all the objects that were involved in the processing, on-the-fly, using new parameters if dependencies and transformational programs have changed. (refer to Figure 4). We can also retrieve parameter values for the matched sources across the different nights. If we suspect any wrong element in the process, we can trace backwards or, more importantly, rerun only that part of the derivation process and compare results. All this data (old and new) is included and becomes part of the system. We demonstrate by use of the Object Viewer (Section V-A) and the Cutout Service (Section V-B) how to view lineage data in the system. Other methods not included in this paper include use of Python scripts or SQL Statements to retrieve lineage data.

A. Object Viewer

The object viewer queries and displays all history processing information for an object from the Astro-WISE database. This tool will show everything related to a derivation of a certain object including attributes, parameters, dependencies and software that were used to create the object. A tree-like view of the object is shown below of all ancestors of a *SourceList* with ID 609571. Leaves in the tree correspond to attributes that have an atomic type and branches correspond to links/references to objects. By expanding a branch one can have a more detailed look at the referenced object. Most information displayed by the object viewer has been left out due to space limitations.

```
Main-Object : SourceList
 |- SLID            | 609571
 |- creation_date   | 2009-05-13 14:30:51
 |- filename        | tmp1242225046.....cat
 |- llDEC           | -0.669179059846
 |- llRA            | 150.083649984
 |- lrDEC           | -0.669552489367
 |- lrRA            | 149.952231788
 |- name            | GAS-Sci-WVRIEND-WFI----
 ---#842-ccd56-Red---Sci-54964....fits.slist
 |-+number_of_sources | 30
 |-+sexparam
 |-+sources
 | +[level : 2] AstrometricParameters  -->
 | | +[level : 3] Chip
 | | +[level : 3] Filter
 | | +[level : 3] Instrument
 | | +[level : 3] ObservingBlock
 | | +[level : 3] PreastromConfig
 | | +[level : 3] AstrometricParameters
 | | +[level : 3] ReducedScienceFrame
 | | | +[level : 4] Astrom
 | | | +[level : 4] BiasFrame
 | | | +[level : 4] Chip
 | | | +[level : 4] ColdPixelMap
 | | | +[level : 4] Filter
 | | | +[level : 4] MasterFlatFrame
 | | | +[level : 4] HotPixelMap
 | | | +[level : 4] IlluminationCorrection
 | | | +[level : 4] Imstat
```



(a) (b) (c) (d) (e) (f)

Figure 5. **The images above show all image cutouts of one of the sources. The (a) RawFrame, (b) ReducedFrame, (c) RegriddedFrame (d) CoaddedRegriddedFrame were used in the Image Pipeline and (e) Illumination Correction and (f) MasterBias were used in the calibration pipelines during the processing of SourceList**

```
[.... some output omited .....]
```

B. Dependency Cut-out Service

This service provides methods to create cutouts of a requested size from the all dependant images (i.e. images generated or used in the processing of an object). The service follows lineage in the system to find all dependant images and the actual location of the object on the dependant images. Figure 5 shows an example of image cutouts of some of the images that were created or used during the processing of one of the sources.

VI. Related Work and Conclusion

Many provenance models exist today, interested readers are referred to [1] and [2] for more details. We have used object oriented design and database support for persistent objects to trace lineage (provide provenance) of data and support e-science research. Although our approach differs from existing models, it does not necessarily replace existing methods, especially for those readers who are interested in only the derivation history of data. We introduce another concept in lineage tracing and use of lineage in e-science that may incite further research in this area. Already applications of provenance data (e.g. [11]) have started to gain attention.

In this paper, we have described the Astro-WISE data lineage Framework. We have shown how Astro-WISE traces lineage of data, how it uses the same data for retrieving, archiving and processing of scientific data which has relevance to e-science data processing needs. Using object persistence, inheritance and polymorphism, users can extend functionality of the system, create, store and distribute links. The target processor, the main processing engine, follows the dependency logic and lineage of data while processing targets. The full end-to-end linking of all dependent data items facilitates this break-through. This abstraction enables Astro-WISE to guide the user to that intrinsic information by forcing full backward and forward chaining in the data modeling. Experimentation has been done using real astronomical examples. These examples demonstrate that lineage information is highly effective in scientific processing and greatly benefits scientific users of the system. Further extensions to Astro-WISE include using lineage for sub-image (re)processing, which involves running part of derivation process on a subsection of an image.

References

[1] Y. L. Simmhan, B. Plale, and D. Gannon, "A survey of data provenance in e-science," *SIGMOD Rec.*, vol. 34, no. 3, pp. 31–36, 2005.

[2] R. Bose and J. Frew, "Lineage retrieval for scientific data processing: a survey," *ACM Comput. Surv.*, vol. 37, no. 1, pp. 1–28, 2005.

[3] C. Scheidegger, D. Koop, E. Santos, H. Vo, S. Callahan, J. Freire, and C. Silva, "Tackling the provenance challenge one layer at a time," *Concurr. Comput. : Pract. Exper.*, vol. 20, no. 5, pp. 473–483, 2008.

[4] M. Greenwood, C. Goble, R. Stevens, J. Zhao, M. Addis, D. Marvin, L. Moreau, and T. Oinn, "Provenance of e-science experiments - experience from bioinformatics," in *Proc. UK e-Science All Hands Meeting 2003*, 2003, pp. 223–226.

[5] M. Zhang, X. Zhang, X. Zhang, and S. Prabhakar, "Tracing lineage beyond relational operators," in *VLDB '07: Proceedings of the 33rd international conference on Very large data bases*. VLDB Endowment, 2007, pp. 1116–1127.

[6] J. Frew, D. Metzger, and P. Slaughter, "Automatic capture and reconstruction of computational provenance," *Concurr. Comput. : Pract. Exper.*, vol. 20, no. 5, pp. 485–496, 2008.

[7] Y. Cui and J. Widom, "Lineage tracing for general data warehouse transformations," in *VLDB '01: Proceedings of the 27th International Conference on Very Large Data Bases*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2001, pp. 471–480.

[8] A. G. Woodruff, "Supporting fine-grained data lineage in a database visualization," Berkeley, CA, USA, Tech. Rep., 1997.

[9] S. B. Davidson and J. Freire, "Provenance and scientific workflows: challenges and opportunities," in *SIGMOD '08: Proceedings of the 2008 ACM SIGMOD international conference on Management of data*. New York, NY, USA: ACM, 2008, pp. 1345–1350.

[10] J. Kim, E. Deelman, Y. Gil, G. Mehta, and V. Ratnakar, "Provenance trails in the wings/pegasus system," in *Concurrency and Computation: Practice and Experience*, vol. 20, no. 5, 2008, pp. 587–597.

[11] F. T. Oliveira, L. Murta, C. Werner, and M. Mattoso, "Using provenance to improve workflow design," pp. 136–143, 2008.