

Analyzing Related Raw Data Files through Dataflows

Vitor Silva, Daniel De Oliveira, Patrick Valduriez, Marta Mattoso

▶ **To cite this version:**

Vitor Silva, Daniel De Oliveira, Patrick Valduriez, Marta Mattoso. Analyzing Related Raw Data Files through Dataflows. Concurrency and Computation: Practice and Experience, 2016, 28 (8), pp.2528-2545. 10.1002/cpe.3616 . lirmm-01181231

HAL Id: lirmm-01181231

https://hal-lirmm.ccsd.cnrs.fr/lirmm-01181231

Submitted on 11 Oct 2018

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Analyzing Related Raw Data Files through Dataflows

Vítor Silva[1,†], Daniel de Oliveira[2,¥], Patrick Valduriez[3,∂] and Marta Mattoso[1,†]

[1]Computer Science, Federal University of Rio de Janeiro / COPPE, Rio de Janeiro, Brazil
[2]Institute of Computing, Fluminense Federal University, Niterói, Brazil
[3]Zenith team, Inria and LIRMM, Montpellier, France

SUMMARY

Computer simulations may ingest and generate high numbers of raw data files. Most of these files follow a *de facto* standard format established by the application domain, *e.g.,* FITS for astronomy. Although these formats are supported by a variety of programming languages, libraries and programs, analyzing thousands or millions of files requires developing specific programs. Database Management Systems (DBMS) are not suited for this, because they require loading the raw data and structuring it, which gets heavy at large-scale. Systems like NoDB, RAW and FastBit, have been proposed to index and query raw data files without the overhead of using a DBMS. However, these solutions are focused on analyzing one single large file instead of several related files. In this case, when related files are produced and required for analysis, the relationship among elements within file contents must be managed manually, with specific programs to access raw data. Thus, this data management may be time-consuming and error-prone. When computer simulations are managed by a Scientific Workflow Management System (SWfMS), they can take advantage of provenance data to relate and analyze raw data files produced during workflow execution. However, SWfMS register provenance at a coarse grain, with limited analysis on elements from raw data files. When the SWfMS is dataflow-aware, it can register provenance data and the relationships among elements of raw data files altogether in a database which is useful to access the contents of a large number of files. In this paper, we propose a dataflow approach for analyzing element data from several related raw data files. Our approach is complementary to the existing single raw data file analysis approaches. We use the Montage workflow from astronomy and a workflow from Oil and Gas domain as I/O intensive case studies. Our experimental results for the Montage workflow explore different types of raw data flows like showing all linear transformations involved in projection simulation programs, considering specific mosaic elements from input repositories. The cost for raw data extraction is approximately 3.7% of the total application execution time.

KEY WORDS: Scientific Workflows; Raw Data File Analysis; Dataflow.

1. INTRODUCTION

Computer simulations have become ubiquitous in scientists' daily duties, once they perform analyses using complex computational models and increasing volumes of data [1]. This allows for the exploration of domain-specific data to support scientists (henceforth named users) in the validation of specific behaviors or scientific hypotheses. Moreover, most large-scale computer simulations involve the execution of many programs that are data-intensive. Typically, each simulation program is characterized by the ingestion and production of large quantities of data. Most of the generated data is stored using several files with heterogeneous formats. Domain-specific areas adopt a common file format, such as FITS [2] in astronomy, or NetCDF [3] and HDF5 [4] in computational fluid dynamics. Each file format is self-described, machine-independent and offers support to a variety of programming languages. They are normally adopted by a large number of libraries and are widely used by reference simulation programs in their domain area. These files are also known as *raw data files* as opposed to structured data ready to be queried by a Database Management System (DBMS). In addition, these files often contain binary or semi-structured data, which makes data analysis difficult.

† E-mail: {silva, marta}@cos.ufrj.br
¥ E-mail: danielcmo@ic.uff.br
∂ E-mail: patrick.valduriez@inria.fr

In computer simulations, the users typically need to validate their hypotheses by analyzing not only the last resulting file, but also its dataflow path. A dataflow in this context may be defined as a composition of data transformations, consuming one or more data sets as input and producing one or more data sets as output. Each *data set* can be defined as any set of *data elements,* which has some predefined *attributes* in each data element. A *data transformation* performs some data processing based on procedures from simulation programs using files as input and output data. Without dataflow support, the main challenge is to analyze data propagation from a large number of data sets (such as files), which are related by the simulation program execution flow. Analyzing these files manually involves naming files accordingly, writing several specific analytical programs and retracing the file transformation flow. This may be tedious and error-prone with high volumes of data. Consequently, it is not trivial to trace data elements related by simulation programs. Let us consider three important types of queries in the exploratory analysis of raw data files, which require access to:

(1) Domain-specific file content;

(2) Multiple files related by simulation programs;

(3) Specific related elements from multiple files.

While queries of type 1 need raw data access to a single file, queries of type 2 or 3 require dataflow access paths. To support queries of type 1, raw data files need to be parsed, following a specific format, and gathering relevant domain-specific data based on the query specification.

Figure 1 shows an example of queries of types 1, 2 and 3 using raw data from the astronomy domain based on the Montage toolkit [5] (Section 4 presents more details of Montage). Suppose that the user wants to perform a domain-specific query on the content of the *projected_images.tbl* file (type 1). Attributes *FA* and *FB* (marked by a gray line in *projected_images.tbl* file) need to be parsed and retrieved to analyze linear transformations from the *projection* program of the simulation with a given configuration. These attributes are also important in tracing anomalies or errors from custom mosaic generation, and can be used to define some conditions for the query. The black arrows represent the sequence of files in *tbl* and *fits* formats, which are projected into another *tbl* file and transformed to create a mosaic in *jpg* format. When these relationships are registered, they allow for queries of type 2, *e.g.,* tracing back the intermediate files that led to *mosaic.jpg* file generation. Finally, Figure 1 further shows data element flows – queries of type 3 – by the gray arrows. In this example, the attribute *CRVAL1* is used as a *key* that relates data elements from different raw data files. Therefore, the file *hdu_1n.fits* can be related to the *FB* element: 0.001969140. Consequently, it is also possible to analyze HDU file content (*i.e.,* attribute *HDU_MATRIX*) in relation to the linear transformations (*e.g.*, attributes *FA* and *FB*). Without dataflow support, users have to write programs that search for each of these *key* values along the *file flow* and write more code to relate and analyze them, for example, in a selected region of the space (*i.e.,* attribute CRVAL1).

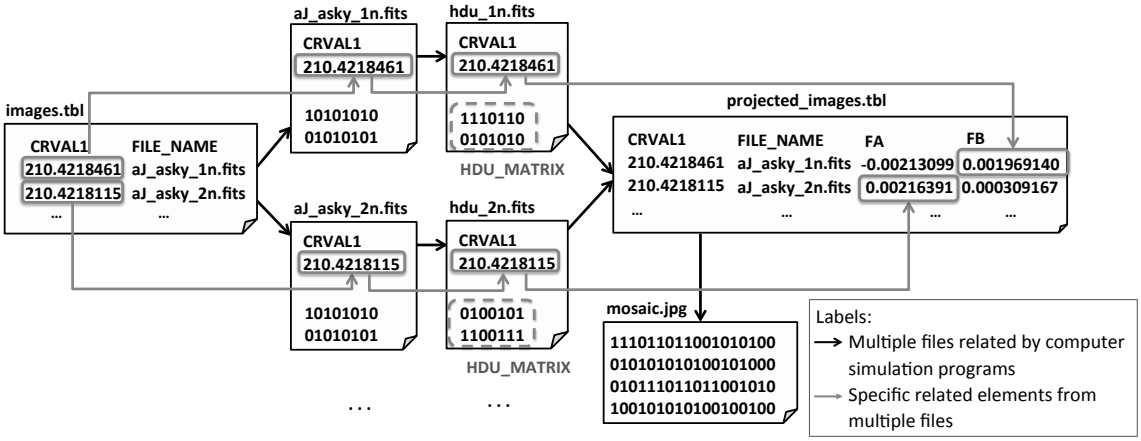


Figure 1. An example from astronomy simulation for the queries in the exploratory analysis of raw data files.

In another example of queries of type 3, users may need to analyze which projected planes (*i.e.,* HDU files) were generated for the mosaic in a specific region on the sky (*e.g.,* this region can be limited by the *CRVAL1* element – floating-point value for positioning reference). Thus, this kind of analysis needs to manage file transformations along simulation programs and extract raw data content from files. Users have to parse the raw data (*i.e.*, domain-specific data) for filtering their results. In this case, one mosaic dataflow may have more than 100,000 files to be parsed.

1.1. Problem Overview

There are approaches that allow for analyzing domain-specific file contents (queries of type 1), such as FastBit [6], FastQuery [7], SDS/Q [8], NoDB [9] and RAW [10]. They all work directly on the raw data file and avoid the overhead of raw data transformation for loading into a DBMS. Even when the DBMS has scientific data structures such as arrays and matrices, raw data files must coexist with the transformed data in the DBMS. This means that after the scientific data has been loaded in a DBMS, raw data will continue to be generated by widely adopted libraries and processed by benchmark programs of the associated scientific domains. These solutions represent a step forward with respect to a DBMS approach. However, they do not support analysis of a dataflow resulting from a sequence of data transformations stored in multiple related files. Thus, they lack support for raw data queries of types 2 and 3.

Supporting queries of type 2 requires following a dataflow of multiple files. As data is generated and transformed by the simulation programs, they create a *file flow*. A file flow represents a dependency relationship describing the order of data (file) transformations. Accessing this file flow is essential in analytical queries that need to trace back the history of data transformations. Manually obtaining the file flow after the simulation, when all the files have been generated is error-prone, since the user usually has to rely on file names to trace the flow. A popular solution is to use Scientific Workflow Management Systems (SWfMS), such as Pegasus [11] and AUDAS [12], to trace file generation. To use a SWfMS, the computer simulation must be modeled as a scientific workflow, defined by a set of activities and their data dependencies. When the scientific workflow finishes, analytical queries may be submitted to a database, which represents the file flow among other workflow execution metadata. However, when SWfMS only consider the file flow management, they track the file derivation history, but disregard domain-specific file contents. Thus, they lack support for raw data queries of type 1 and 3.

SWfMS such as Kepler [13] and Panda [14] have specific data structures to support data element and collection flow, but they are not aware of domain-data file content, which also limits the power of supporting queries of type 1 and 3. In the Chiron SWfMS [15], raw data can be extracted from the file contents and stored in its relational database, which captures information about workflow modeling and execution, besides the domain-specific data. Thus, Chiron supports queries of type 1 and 2, which allows accessing the contents of individual raw data files among a large number of files spread in several workspaces (*i.e.*, directories where files are stored). However, Chiron does not support data element flow management, which is the basis for queries of type 3.

Supporting all three types of analytical queries is an open, yet important, problem in a large-scale data-intensive scenario. Solutions for tracking data element and data collection transformations require an engine to manage the dataflow at different granularities, with domain-specific file content support.

1.2. Contributions

In this paper, we extend the dataflow approach of Chiron to introduce the extraction of domain-specific data element along its derivation path. This extraction allows for analyzing multiple-related raw data files through dataflows, thus supporting queries of type 1, 2 and 3. The users may choose domain attributes to be traced and while the computer simulation is running, attribute values are gathered and their data element flow is captured in a DBMS. Users can submit queries for selecting raw data from the registered dataflow during and after the execution of computer simulation. The goal is to get the data elements of interest at the time they are being generated and relate them to other data elements from the same file, as well as to data elements from other related files, produced during the file flow. This avoids parsing the raw data file to find the desired data elements. Also, this data element is available for queries as soon as it is generated, which empowers workflow steering [16].

We incorporate domain data extraction within the process of workflow provenance registry [17]. Thus, the same database relates domain data elements with performance data and coarse grain provenance data. Such data integration does not need data conversions and allows for powerful queries, as it is shown with the astronomy case study based on the Montage workflow [5] and another study based on an Oil and Gas workflow. Our experimental results for Montage workflow in providing different types of raw data access show the power of analytical queries and the cost for this raw data extraction is approximately 3.7% of the total workflow execution time.

This paper is a major extension of the paper presented at the Workshop on Parallel and Distributed Computing for Big Data Applications (WPBA 2014) [18], which introduces the initial idea of the proposed approach without an experimental evaluation. In this paper, we extend [18] by presenting a formal definition for the dataflow concept; dataflow management techniques separated on two abstraction levels; and several

experiments, which evaluate these new ideas including measuring the costs of data extraction using workflows from the astronomy domain and oil & gas.

The rest of the paper is organized as follows. Section 2 discusses related work. Section 3 presents our formalization concepts of dataflow management. Moreover, this section discusses how provenance can be used to access raw data in data-intensive computer simulations. This section also describes raw data extraction and analysis from domain-specific files using Chiron SWfMS. Section 4 presents two data-intensive case studies: a Montage workflow and a synthetic workflow based on Oil and Gas applications. Section 5 presents our experimental evaluation. Section 6 concludes.

2. RELATED WORK

In this section, we discuss the main approaches for raw data analysis for single file and the SWfMS solutions that involve dataflow management at the physical and logical levels. These three approaches correspond to supporting queries of types 1, 2 and 3 respectively. However, to the best of authors' knowledge, there is no other work that allows for combining these three types of analytical dataflow queries.

2.1. Raw data analysis for single file

Several approaches deal with raw data analysis for single files [6,8,9,19,20]. Typically, they access specific data from the files, then parse and index it using specific query languages, engines and APIs, thus relieving the users to develop their own code for each type of analysis. Examples of these approaches are FastBit [6] and Attribute-based Unified Data Access Service (AUDAS) [20]. Other approaches such as FastQuery [7] and SDS/Q [8] deal with indexing and querying data in parallel. SDS/Q processes queries directly over the files, but is restricted to the HDF5 file format [4]. Thus, it eliminates overheads related to data transformation and loading into a different data structure. Its main advantage is to exploit increasing memory capacities, since it uses bitmap indexing and in-memory query processing. It also exhibits performance improvements through parallel query execution, with techniques that parallelize execution across nodes or within a node, and the use of FastBit indexes to improve the performance of semi-join operations.

Alagiannis *et al.* [9] propose NoDB for extracting domain-specific contents from raw data files and populating them in a modified version of PostgreSQL, called PostgresRaw. NoDB avoids data transformations, since it allows for adaptive query processing and improves query performance using statistics and caching strategies. The limitation of NoDB is that raw data is stored in PostgresRaw and the raw data file itself. To overcome these limitations, Karpathiotakis *et al.* [10] propose RAW, a flexible query engine that adapts the query plan to the formats of raw data, without the overhead of loading raw data into another repository or a DBMS. This approach is similar to SDS/Q, yet not specific to one file format. RAW implements Just-in-Time (JIT) access paths and column shreds. JIT access paths are generated at runtime in order to map existing scan operators for the raw data. Also, columns shreds create subsets of columns for some of the data elements or data collections.

None of these approaches allow for dataflow analysis. They focus on "isolated" files and do not consider the dataflow transformations that led to the "final" file that is being indexed and queried. To support dataflow analysis, these approaches need to introduce the physical and logical levels for dataflow management.

2.2. Dataflow management at the physical level

Dataflow management at the physical level designates approaches that support data transformations on the file system, thus ignoring domain-specific file contents. It treats files as black-boxes, *i.e.,* there is no index or query support for domain-specific contents from raw data files. Vahi *et al.* [11] propose two approaches based on data stores, where users do not need to modify anything in their programs to store the data products in these data stores. The first approach considers object stores to gather all files consumed and produced by a workflow. This solution provides a way to manage the files generated at runtime using a specific indexing scheme to perform queries during or after scientific workflow execution. The second approach, also in [11], performs an evaluation based on a shared file system to gather the files in one or more object stores. Their experiments on big data workflows in a cloud environments (Amazon EC2) [21], considering an I/O-intensive workflow (*i.e.,* Montage workflow) and another CPU-intensive workflow (*i.e.,* Rosetta workflow), show better performance for a shared file system in comparison with a non-shared file system.

The AWARD framework [12] provides a dataflow-based approach, with a tuple-oriented representation to manage experiment execution. This framework supports data distribution in different data centers (different regions). To control dataflow execution, AWARD gathers tuples (a set of parameter values or data

elements, in our approach) at runtime and generates tokens to propagate data between dependent activities. However, AWARD does not consider data elements, such as parameter values in AWARD relations. Furthermore, it is not easy to integrate AWARD with provenance databases, since the AWARD schema is fixed and any modification to insert new metadata would yield major modifications in the AWARD engine.

2.3. Dataflow management at the logical level

SWfMS such as Kepler [13], Panda [14] and Chiron [15] focus on dataflow management at the logical level. However, these approaches are not aware of element flow within raw data files. This limits the analytical power of raw data analysis. Kepler formally defines the chaining of activities in a workflow, representing dataflows as a set of data elements and data collections access. However, these dataflows are limited to data directly manipulated by the workflow engine, *e.g.,* parameter sweep, and not raw data file contents. Thus, Kepler does not support queries of type 1 and has limited support for queries of type 3. Panda proposes an elegant and powerful provenance data formalism to trace file flow and data element / collection flow generated by a workflow composed of relational algebra operations. However, it does not support raw data file access. Similarly, current solutions for provenance, such as Pegasus Lite [11] and Wings [22] do not address dataflow queries as we show in this paper. Chiron implements a data-centric workflow algebra. Workflows are represented and executed as algebraic expressions composed of operations, which encapsulate the simulation programs and operands to represent the dataflow. Its algebraic workflow engine gathers domain raw data and combines it to the dataflow information that is stored in relations of its provenance database. Chiron supports queries of types 1 and 2. However, data element flows are complex to be followed and queries of type 3 are not currently supported.

In this paper, we extend Chiron, to explicitly provide for *data element flows* obtained from file contents. Our approach is complementary to the powerful indexes and query mechanisms for raw data analysis for single files in approaches such as RAW [10].

3. DATAFLOW MANAGEMENT IN RAW DATA FILES

Our approach for managing dataflows involving raw data files is based on the data-centric relational algebraic approach previously proposed by our team in [15, 23]. As in any algebra, data is represented as operands and transformations are ruled by operations. In the next section, we review definitions for this algebra from [23] and show how it may represent the dataflow including element flows. In particular, we show how operand relations represent domain-specific data extracted from raw data files. In Section 3.2 we show how the algebraic notation for dataflows is represented in a provenance database schema.

3.1. Algebraic notation for dataflows

We provide a dataflow definition, which is used in the remaining of this paper. We assume that each *data set* is any set of *data elements,* which has some predefined *attributes* in each data element. In a dataflow scenario, a *data transformation* performs some data processing based on some procedures from simulation programs, consuming data from one or more *data sets* (*i.e.,* inputs) and producing one or more *data sets* (*i.e.,* outputs).

Let T_1 and T_2 be two data transformations. The composition $T_1 \circ T_2$ is a transformation that first applies T_1 to input data sets I_1 to obtain intermediate data sets I_2. It then applies T_2 to I_2 to obtain output data set I_3. Composition is associative, so we denote the linear composition of n data transformations as $T_1 \circ T_2 \circ ... \circ T_n$. Therefore, we define such composition of n data transformations as a dataflow D_F, which can be represented as $D_F = T_1 \circ T_2 \circ ... \circ T_n$. The output data sets I_{n+1} from this dataflow can be represented as $I_{n+1} = (T_1 \circ T_2 \circ ... \circ T_n)(I_1)$, or $I_{n+1} = (D_F)(I_1)$ for short, where I_1 represents the input data set of this dataflow. This definition was adapted from the *workflow instance* concept presented by Ikeda *et al.* [24].

Dataflows can be represented using a data-centric workflow algebra [23]. The main advantage is to add semantics to the data transformations. For example, the algebra from [23] has a set of operators (Map, Reduce, Filter, SplitMap, MRQuery and SRQuery) to rule data transformations (*i.e.,* workflow activities). If T_i takes a data element from its I_i and transforms it into one corresponding data element in I_{i+1} we may say that T_i has the behavior of a Map operator. If T_i generates an I_{i+1} that is a subset of I_i, we may say that this T_i behaves like a Filter operator. Each T_i of a D_F is associated to an operator, where its input data set I_i is represented as an operand input relation(s) and produces an output data set I_{i+1} that is also represented as a relation. Likewise, an output relation can also be the input relation to the operator associated to the next data transformation. For each T_i, the user must specify its input data sets I_i, and output data sets I_{i+1}, which are mapped to relations (composed by a set of attributes and their associated values, *i.e.,* parameter values). An algebraic data transformation can be represented as the expression:

$$I_{i+1} \leftarrow Operator\ (\ T_i,\ Optional\ operands,\ I_i\ )$$

A dataflow is then represented as a set of algebraic expressions. The relations consumed and produced by T_i can be represented as a dependency graph, which is useful to represent the entire dataflow processing. In each relation that represents I_i, each tuple corresponds to a data element consumed or produced by T_i.

Let us consider a simple dataflow with two data transformations, T_1 and T_2, as illustrated in Figure 2. The boxes represent program invocations responsible for data transformations and <<*stereotypes*>> represent algebraic operators that describe the data transformation behavior. T_1 is executed *n* times with the following input data set configuration: (1) *Param1*: a binary input file; (2) *Param2*: a numerical attribute value; and (3) *Param3*: a numerical threshold to be used by T_1. Each of the *n* executions of T_1 consumes one DAT file and produces one BIN file (*i.e.,* it corresponds to *Param4* value) as output. Both the DAT and BIN files may contain important information that users need for analysis. T_1 is executed for each data element of data set *R*, since the Map operator rules this data transformation. For each execution, one resulting data element is generated in the output data set *S*. Then, T_2 invokes another program execution that consumes *S*, aggregating new data according to the *Param3* value. When one execution of data transformation T_2 produces another BIN file, which corresponds to *OutputFile* value, the user might be interested in tracking a specific data element from the contents of each of these *OutputFile* values. This would be the moment where a new operation would be inserted into the dataflow to invoke a program that would open the BIN file from attribute *OutputFile*, extract the necessary information and insert this extracted value in the corresponding attribute from relation *V* (*i.e.,* attribute *Result*). In this case, the extraction process is represented as raw data gathering from file *hdu_1n.fits*. This extraction enriches the operand relations with domain-specific data in order to empower raw data analysis.

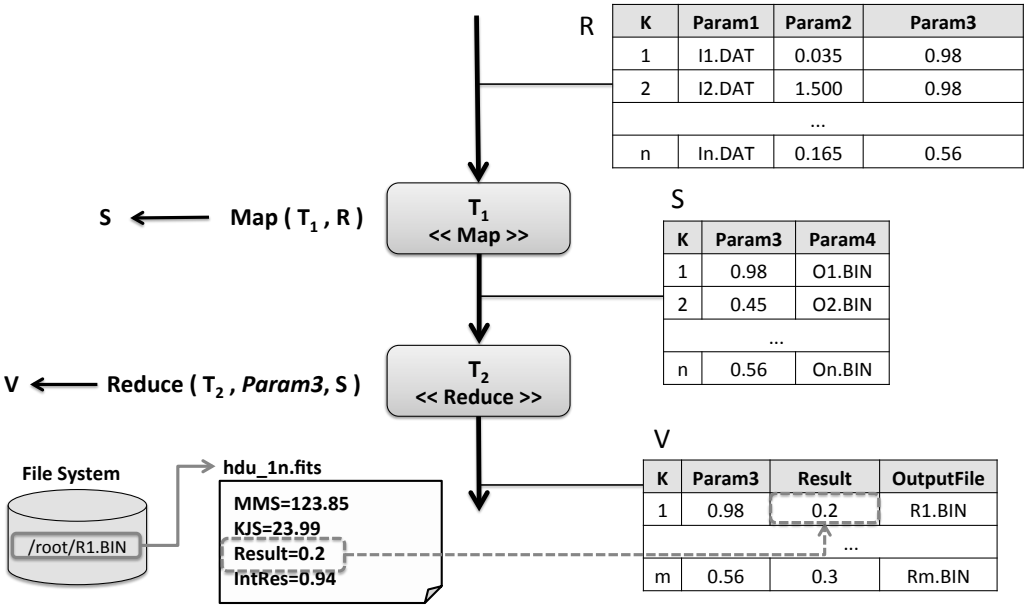


Figure 2. Example of the algebraic representation of dataflow with raw data elements.

3.2. Managing dataflows in provenance databases

Using relations to represent data consumed and produced by data transformations allows for representing the dataflow from computer simulations uniformly in a provenance database. The relational approach allows for querying raw data in a structured way, thus easing user's analysis. Chiron's engine is driven by the algebra expressions, which are stored in its provenance database. Thus, Chiron's provenance database stores dataflow specifications following operators of the algebra expressions (prospective provenance) and the metadata associated to dataflow execution (retrospective provenance) [17], all in the same database. In other words, prospective provenance captures the dataflow specification (such as data transformations and the flow of data sets), while retrospective provenance determines the properties about dataflow execution (such as the directory to execute a computer simulation or the elapsed time to complete a data transformation) [17].

Since in this paper, we consider file flow and data element flow management, we extend Chiron's provenance database to store metadata related to domain-specific data, represented as the consumed and produced files, and their data elements, extracted from raw data files (*i.e.,* files in scientific file format).

A reference model for representing provenance of a scientific workflow named PROV-Wf has been

proposed [25] and employed in Chiron. PROV-Wf is a specialization of the PROV W3C recommendation [26], to accommodate provenance data specific to scientific workflows, in particular, large-scale parallel executions. Considering the dataflow definition given in Section 3.1, we extended/adapted PROV-Wf to support dataflow concepts. This extended/adapted provenance data model, named PROV-Df, is shown in Figure 3. PROV-Df is composed of three parts: the structure of the dataflow (white classes in the UML class diagram), execution of the dataflow (dark gray classes) and environment configuration (light gray classes). Each class in PROV-Df is extended from a PROV component. The stereotypes in the UML class diagram are used to represent PROV components. The white classes represent the dataflow specification (entity *Dataflow*), describing the data transformations (entity *DataTransformation*), data sets (entity *DataSetSchema*), attributes (entity *Attribute*) and types of manipulated files (entity *FileType*). In this paper, we add a representation for data dependencies between data transformations in PROV-Df, which can be represented in this data model by the *WasAssociatedWith* relationship between entities *DataTransformation* and *DataSetSchema*.

The dark gray classes express information about dataflow execution (activity *ExecuteDataflow*), considering properties from the execution of data transformation (activity *ExecuteDataTransformation*), data set instantiation (entity *DataSet*), data elements from a data set (entity *DataElement*), establishments of attribute value to a data element (entity *AttributeValue*) and the program invoked by a data transformation (software agent *Program*). Furthermore, the light gray classes provide information about the computing environment. In this data model, agent *Machine* represents the computational resources used to execute a dataflow, and agent *Scientist* corresponds to the users involved in the dataflow specification. We further adjust some relationships between *WActivity*, *RelationSchema*, *ExecuteActivity*, *Field, Relation* and *Value* in PROV-Wf from [25] (*i.e.*, they are equivalent to the relationships between *DataTransformation, DataSetSchema*, *Attribute*, *DataSet*, *DataElement* and *AttributeValue* in PROV-Df) to allow for dataflow management at the physical and logical levels. These modifications aim at representing data consumption and production in a dataflow point of view (*i.e.*, data transformations and data dependencies).

PROV-Df classes are mapped to tables in the provenance database. For each data set consumed and produced in the dataflow, we create a physical table in the chosen DBMS with associated attributes and values. Each table is responsible for storing domain-specific data at runtime.

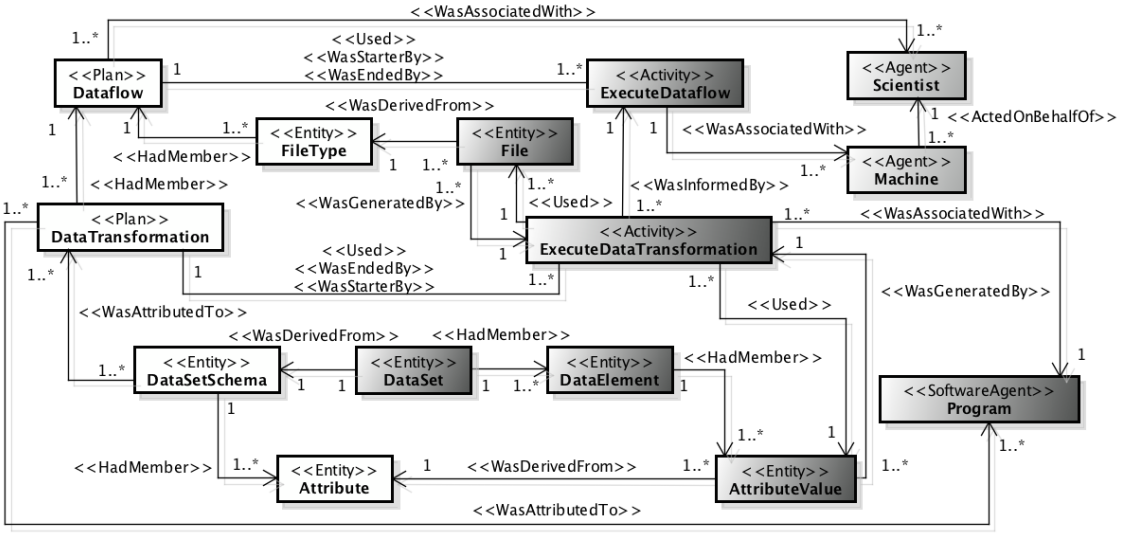


Figure 3. PROV-Df data model [25].

Considering the characteristics of the PROV-Df model and the new version of Chiron's provenance database, the users may query consumed and produced data while submitting analytical queries of types 1, 2 and 3. Thus, users can query the provenance repository to gather information about the entire dataflow and its execution (such as start time, end time and errors from the execution of a data transformation). Provenance databases provide access to selected raw data files among a large number of generated files spread in several workspaces (*i.e.*, directories in different machines). The reference to files is registered in the provenance database as pointers (URI), which are also related to the dataflow and domain-specific data values (that are actually contents from those files). Thus, users can elaborate analytical queries on the provenance database, which involve domain-specific data and dataflow information to identify files and data elements within these files to perform deeper analysis.

To illustrate the potential of provenance data to access and query files and their domain-specific content (*i.e.,* raw data), let us consider again the dataflow from Figure 2. A user may want to know which files are produced by the execution where the value of *Param1* is 0.035 and the *Result* is 0.2. Depending on the size of the computer simulation, manual querying of this data is infeasible since users have to track and analyze all the files generated during the execution of the computer simulation and find a way to relate them through their contents. The usage of a dataflow-aware provenance database enables such type of query. However, to avoid potential bottlenecks when considering parallel execution, we modify the relationship between *DataTransformation* plan and *DataSetSchema* entity in order to specify data dependency between data transformations. It is important to highlight that this modification in this dataflow provenance database allows performing queries of type 3 and improves the elapsed time for running queries of type 2.

Based on the PROV-Df data model, our approach supports both file flow and data element flow management using a single provenance database. We implemented an extended version of Chiron, which uses a provenance database that follows the PROV-Df with the aforementioned modifications. Using Chiron, users can specify the data dependencies among data transformations, invoke programs for each data transformation, access raw data from files and analyze these data using the provenance database at runtime.

3.3. Raw data extraction in dataflows

To model a dataflow using Chiron, the users must define the data transformations and their data dependencies. Then, the users specify a *program invocation* for performing each data transformation. Considering the dataflow execution, it is important to mention that before each program invocation, Chiron's parallel engine instruments program invocation replacing some labels (*i.e.*, each label represent the name of an attribute) by attribute values (known as *instrumentation process*). In the dataflow specification, after a data transformation, the users may further add a program invocation (*i.e.* with their own code) to extract domain-specific data from files (known as *extraction process*). These programs are named *extractors*, which can be defined as an optional post-processing step for each data transformation. Therefore, the execution of a data transformation in Chiron yields the execution of these three steps: instrumentation, program invocation and extraction.

Raw data extraction using Chiron requires the invocation of a user program, which has to access and gather relevant domain-specific data from files according to the data elements specified in the modeled dataflow. The *raw data access* can be developed from scratch, where users can use built-in language programming libraries. However, depending on the file format, such as binary files, data access is not trivial and this task is laborious, tedious and error-prone. Thus, it is more appropriate to use specific libraries for these file formats, such as ROOT [27], a data analysis framework, and HDF5DotNet [4]. Moreover, users must define the attributes to be gathered (*i.e.,* data elements) in a tabular format (similar to the CSV file format with delimiter semicolon), known as *ERelation* file, using some developed tools. In this file, the first line has the attribute names and the remaining lines contain the attribute values (*i.e.,* gathered data elements). Chiron gathers data from all *ERelation* files generated by the executions of a data transformation (also known as *activations*) and stores these data in a physical table (*i.e.*, provenance database). Thus, user's developed tools are responsible for accessing and gathering raw data from files, while Chiron's engine automatically creates relations for the domain data, relates them to the provenance model and finally inserts these data in the provenance database. The extraction execution time depends on the tool used to gather raw data from files as well as the volume of extracted data. The data extraction occurs at the same computing node that produced the raw data file and the corresponding upload at the provenance database is done through insert operations using several nodes.

4. DATAFLOW CASE STUDIES

In this section, we introduce two dataflows case studies used in our experimental evaluation (Section 5) and in the examples provided throughout this paper: the Montage dataflow in astronomy and an Oil and Gas dataflow. They are modeled using the Chiron SWfMS and executed in a computer cluster. Using Chiron's algebraic approach, we first modeled the Montage dataflow using an astronomy toolkit named as Montage [28]. This toolkit is used for assembling astronomical images into custom mosaics, suitable for large scale processing of the sky. Montage provides to astronomers a service to build mosaics in Flexible Image Transport System (FITS) format [2], according to common astronomy coordinate systems, arbitrary image sizes and rotations, and all World Coordinate System (WCS) map projections. Thus, Montage uses different astronomical images to assemble them into custom mosaics, respecting some universal formats.

The Montage dataflow is composed of nine data transformations as shown in Figure 4. The boxes represent data transformations and <<*stereotypes*>> represent algebraic operators that rule the data transformations. Sometimes, the input data set of a data transformation is a projection of the attributes from

the output data set of preceding data transformation (*i.e.,* a subset of the pre-defined attributes). The first data transformation (*List FITS*) extracts several FITS files from a compressed file (obtained from an external astronomy repository - 2MASS [29]). Each input FITS file has 20 types of domain-specific data, which are defined as attributes in the data sets in Chiron's algebra [23]. The second data transformation (*Projection*) computes the projection of these astronomy-positioning references into a specific plane (extraction of 2 attributes and propagation of 19 previous attributes). Then, *Select Projections* joins FITS projection files that are associated to the same mosaic (extraction of 2 attributes). *Create Uncorrected Mosaic* creates a mosaic without overlap interferences and color corrections and, as a result, it creates a JPG image (extraction of 1 attribute, the JPG file). The other data transformations from the Montage dataflow are defined to consider overlap interferences and color corrections in order to create a corrected custom mosaic.

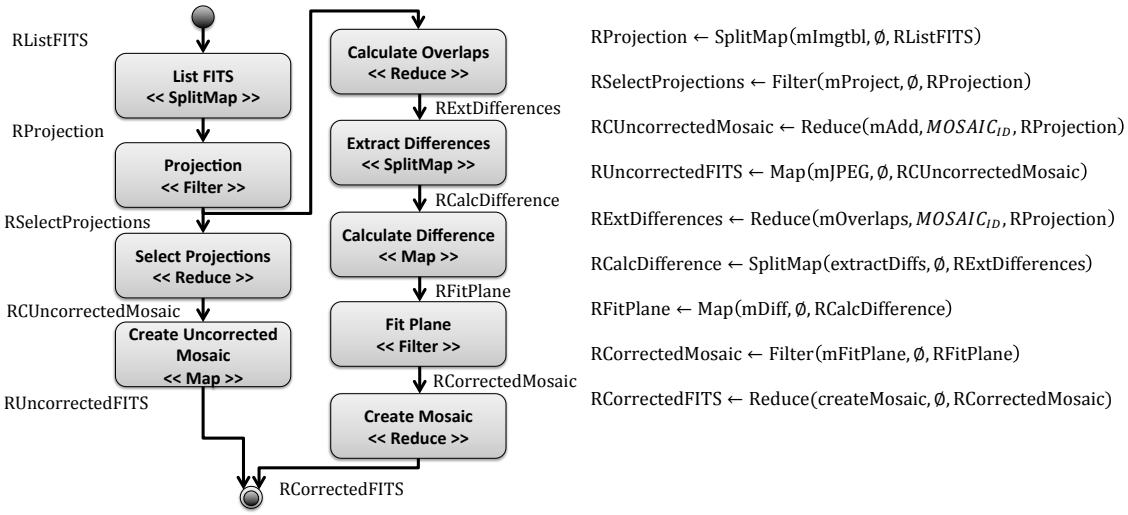


Figure 4. The Montage dataflow specification with associated algebraic operations and operands.

In Figure 5, we simplify Montage to show the algebraic relations – or data sets – only for some representative data transformations (programs), *List FITS* and *Projection*. Data transformation *List FITS* consumes the mosaic identifier (*MOSAIC_ID*) and file *rep1.tar.gz* (*REPOSITORY*) as input data set (represented as *RListFITS*). For each file in this compressed file, the data transformation *List FITS* creates a new data element in the output data set (known as *RProjection*). Each data element contains the mosaic identifier (*MOSAIC_ID*), a FITS file identifier (*CNTR*) (in order to trace back) and 2 extracted elements (*CRVAL1* and *CRVAL2*) that represent two coordinate values to determine a position in the native image coordinate system (*e.g.,* RA, Dec). Note that these extracted attribute values are within files. Then, data transformation *Projection* processes each data element of the data set *RProjection* and for each one, extracts 2 new attribute values from raw data files (*i.e., HDU_AREA*, which records the region covered in output pixel space by the input file to generate a mosaic, and *HDU_FILE*, which records the reprojected FITS file after a projection to a specific mosaic configuration). To extract these attribute values, we developed an extractor program, named *extractDiffs*, which is invoked during dataflow execution. There are other attributes that are extracted after the execution of data transformation *Projection*, to allow users to analyze the raw data.

Due to the huge volume of raw data files produced during Montage dataflow execution, it would be impossible to analyze such raw data without the support of the provenance repository. In Section 5.2, we present an experimental evaluation of the Montage dataflow execution with and without raw data extraction. Furthermore, we show how provenance helps raw data file and dataflow (*i.e.*, file flow and data element flow) analyses using this dataflow. We highlight that the Montage dataflow in real execution does not need raw data extraction, since the dataflow is normally defined by the chaining of components (*i.e.*, programs) from the Montage toolkit. Therefore, the Montage dataflow modeling without raw data extraction can present the same data transformations from Figure 4, however it only considers the relationship between the consumed and produced files by each astronomy program. In this paper, we present raw data extraction as a contribution for managing dataflow execution, which is an optional feature while users are modeling dataflows. Consequently, the extracted raw data are stored in our provenance database.

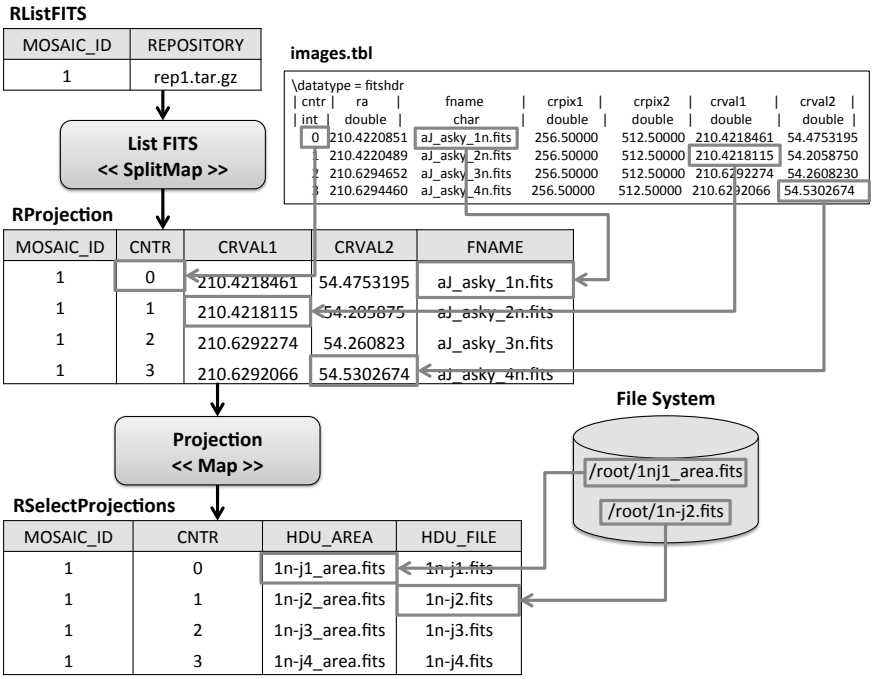
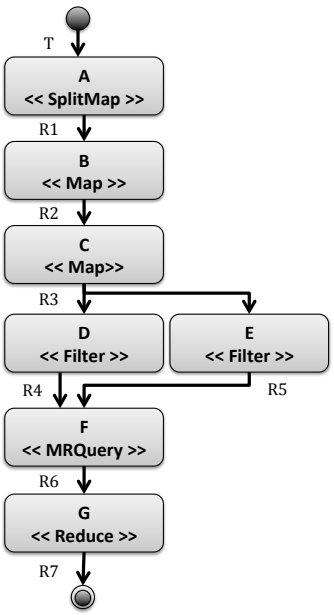


Figure 5. An excerpt of the raw data indexing using provenance for the Montage dataflow

The second case study is a synthetic dataflow whose execution behavior is based on an Oil and Gas simulation [23]. This synthetic dataflow was developed using a Scientific Workflow Benchmark (SWB) [30] where data transformations are configured to present an equivalent behavior to the real computer simulation, as presented in Figure 6. This synthetic dataflow has seven data transformations, which present different algebraic operators implemented by Chiron. It also performs raw data extraction for most of the data transformations. Although it is not a real execution of the Oil and Gas dataflow, the file production is emulated by SWB, thus allowing for data extraction.



$$R1 \leftarrow \text{SplitMap}(A, \emptyset, T)$$
$$R2 \leftarrow \text{Map}(B, \emptyset, R1)$$
$$R3 \leftarrow \text{Map}(C, \emptyset, R2)$$
$$R4 \leftarrow \text{Filter}(D, \emptyset, R3)$$
$$R5 \leftarrow \text{Filter}(E, \emptyset, R3)$$
$$R6 \leftarrow \text{MRQuery}(SQL_{query}, \emptyset, \{R4, R5\})$$
$$R7 \leftarrow \text{Reduce}(G, REDUCE_{VALUE}, R6)$$

Figure 6. The synthetic dataflow specification with associated dataflow in the algebra expressions.

The dataflow starts its execution in data transformation *A*, which is ruled by the SplitMap operator. Therefore, the input dataset has to define a split factor [30], which can be described by the number of data elements produced after the consumption of an input data element for this data transformation. In our experiments, we fix the value of the split factor as 240. Then, the dataflow has two data transformations (*B* and *C*) executed in sequence and ruled by a Map operator. Since data transformation *C* is executed completely, two other data transformations (*D* and *E*) are invoked in parallel. The Filter operator rules these data transformations, which require two filter factors. We consider a filter selectivity of 60% and 80%,

respectively. Once data transformations *D* and *E* have finished their execution, the data transformation *F* (ruled by MRQuery operator) joins the data elements produced by the previous data transformations. This data transformation specifies SQL_{query} by the respective query specification: *select R4.ID, R4.REDUCE, R4.REDUCEVALUE from R4, R5 where R4.ID = R5.ID*. Finally, data transformation *G* is ruled by the Reduce operator with $REDUCE_{VALUE}$ as an aggregation attribute.

5. EXPERIMENTAL EVALUATION

In this section, we evaluate our approach for analyzing raw data through dataflow. Our experiments are based on the two dataflows (*i.e.*, Montage and Oil and Gas Extraction) introduced in Section 4. In these experiments, we measure the performance with and without raw data analysis support.

5.1. Environment Setup

We use a SGI Altix ICE 8400 cluster of the High Performance Computing Center - NACAD/UFRJ [31]. This cluster has 1.28 TBytes of distributed RAM memory and 72 TBytes of shared disk under a Suse Linux Enterprise Server operating system. The network is configured using QDR and DDR Infiniband switches. This computing environment also consists of two separated job queues. In our experiment executions, we consider the job queue with 64 CPUs Quad Core Intel Xeon X5355 (Clovertown), 2.66 GHz (256 Cores).

5.2. Experimental Results

We performed three experimental evaluations with the following goals:

- **Extraction cost (named as COST)**: Measurement of the cost for instrumenting and extracting raw data using the Montage dataflow. The extraction cost considers the elapsed time to process a raw data file, extract the specific information and store this data in the provenance database;
- **Workload analysis for raw data extraction (named as WORKLOAD)**: Evaluation of the raw data extraction cost using the Oil and Gas dataflow with different workloads. In this situation, we consider two analyses that evaluate different execution times for data transformation following a Gamma distribution [30], and dataflow execution varying the size of raw data files;
- **Query Processing (named as QUERY)**: Analyses of the raw data query processing support. We present examples to analyze domain-specific file contents; multiple files related by simulation programs and specific related elements from multiple raw data files.

We now present each evaluation in more detail.

COST. We modeled and executed the Montage dataflow with 1,585 FIT files downloaded from the Two Micron All Sky Survey [29]. The execution lasted approximately for 2 hours and 30 minutes using 96 cores on the Uranus cluster. This dataflow represents a data-intensive execution since it manages almost 166,000 files and 500 gigabytes of data. In this evaluation, two versions of the Montage dataflow were developed and executed as presented in Figure 7. The first version is modeled without extractors, *i.e.,* Chiron only manages the file flow. The second version considers the raw data extraction for each data transformation, since we aim at computing the cost of the extraction process using Chiron, *i.e.,* the cost to manage the data element flow. For each version of the Montage dataflow, we consider the average elapsed time of three dataflow executions. The Montage dataflow without extractors ran for 137.85 minutes, while the same dataflow with extractors ran for 143.15 minutes. When executing the Montage dataflow, extracting and indexing the content of files through the provenance database it yields an increase of 3.84% in total elapsed time.

Considering these results, we note that the raw data extraction represents 3.7% (5.3 minutes) of the total elapsed time for the Montage dataflow execution with extractors. This is the time necessary to invoke all extractors (third party programs), gather raw data from files and load it into the provenance database. If we use a complementary solution (such as FastBit) coupled to Chiron, extraction cost may be close to that for parsing and building a representation of the raw data. Furthermore, the extraction cost pays off with some advantages, since the user is able to analyze domain-specific data in this domain-data provenance database and correlate this data with information about dataflow execution and the dataflow transformations at runtime. The query processing advantages are discussed in more detail in the QUERY evaluation.

One of the main difficulties in scientific dataflow executions is fine-tuning the configuration of the dataflow, such as parameters and programs invoked for data transformations. For example, in Montage astronomers often need to adjust input data elements and parameter values from the Montage computer simulations in order to improve the result quality (*i.e.,* custom mosaic image). This adjustment in the input

data elements increases the complexity of the simulation programs to be performed and thus the dataflow elapsed time, since the chosen programs may require more computations to guarantee a better precision. Thus, depending on the configuration of the dataflow, the percentage of the elapsed time associated to the raw data extraction in comparison with the dataflow total elapsed time may decrease, since raw data extraction elapsed time may decrease as less extraction data is needed to fine tune the dataflow. We further evaluate the extraction cost of our solution, when we increase the size of raw data files using the synthetic Oil & Gas dataflow on the WORKLOAD evaluation.

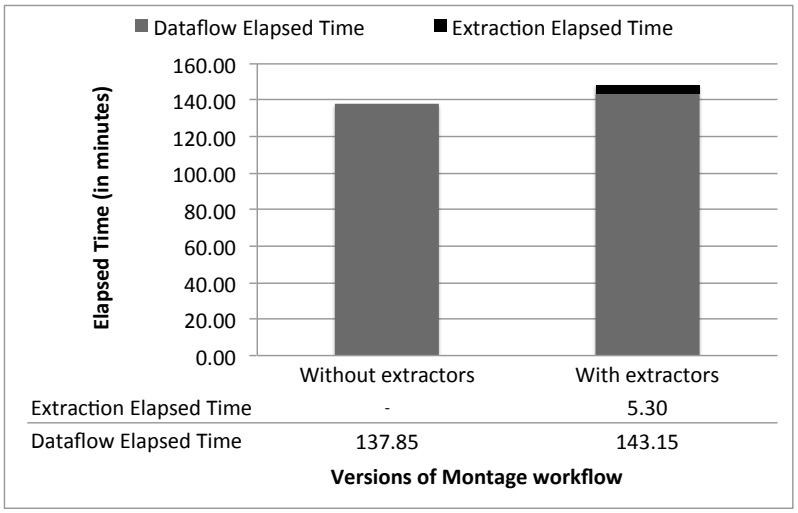


Figure 7. Montage dataflow executions without and with extractors.

WORKLOAD. In this experiment, we use the Oil and Gas Extraction dataflow to evaluate the extraction cost, while varying the complexity to execute a data transformation and the size of raw data files. As discussed in Section 4, we fix the values of the split and filter factors. In the first analysis of this experiment, we only modify the Data Transformation Cost Factor (DTCF), which represents the average elapsed time for each data transformation execution. According to the defined DTCF, the execution time follows a Gamma distribution $\Gamma(\kappa, \theta)$, where $\kappa = 2^{DTCF}$ and $\theta = 1$, for DTCF ≥ 0.

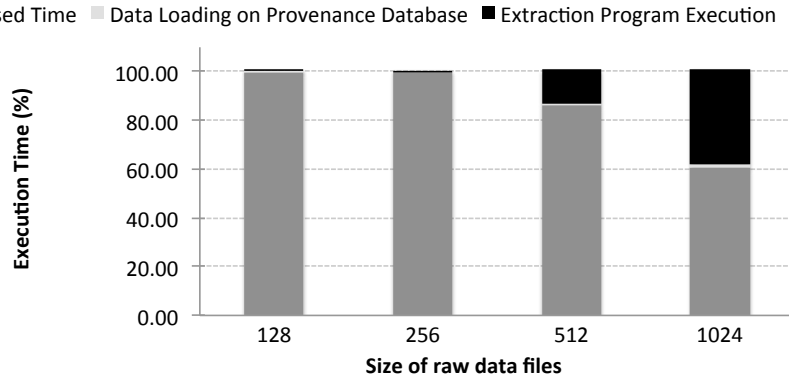
We consider the following values for DTCF: 5, 7 and 8. Table 1 gives the experimental results for the execution of each DTCF configuration using 96 cores. As dataflow elapsed time, we consider the average elapsed time and standard deviation of three dataflow executions. We also ensure a confidence level higher than 95% for dataflow elapsed time. In these executions, this dataflow was submitted in the same conditions, while extraction yield a similar elapsed time for all dataflow executions with a specific configuration. In addition, extraction elapsed time considers the computational processing time for executing extractor programs and loading raw data into the provenance database. In Table 1, we observed that the highest extraction elapsed time occurs when we are using DTCF = 5, since its elapsed time is the lowest of all DTCF configurations. Moreover, as we increase the DTCF value (*i.e.*, we increase the single data transformation elapsed time), we also increase the total dataflow elapsed time. This way, if we also consider the same raw data extraction in relation to computational processing efforts, its elapsed time becomes negligible as the dataflow elapsed time increases, but the output data is the same (*i.e.,* we consider the same raw data extracted from files). For every configuration associated to a compute-intensive dataflow (*i.e.*, with high execution time for data transformations), it is possible to note that the extraction elapsed time is less than 0.50%.

The second analysis also considers the Oil and Gas Extraction dataflow. However, we define a fixed DTCF value for every execution (*i.e.*, DTCF = 6) and vary the size of raw data files, in order to evaluate extraction cost in the presence of larger raw data files. Thereby, we only modify the File Size Factor (FSF) in each dataflow execution, which represents the size of generated raw data files during dataflow execution. According to the defined FSF, the size of raw data files (∂) changes as follows: $\partial = 2^{FSF}$, for FSF ≥ 0. We consider the following values for FSF: 7, 8, 9 and 10 using 64 cores on Uranus cluster. Figure 8 presents the execution elapsed time in percentage for the dataflow execution (*i.e.,* dark gray bars) and the extraction cost that is composed by two aspects: the elapsed time to invoke the extraction programs (*i.e.,* black bars), and the data loading on the provenance database (*i.e.*, light gray bars).

Table 1. Oil and Gas Extraction dataflow executions varying data transformation cost.

DTCF	Single Data Transformation Elapsed time – Average (in seconds)	Average Dataflow Elapsed Time (in minutes)	Standard Deviation for Dataflow Elapsed Time (in minutes)	Extraction Elapsed Time (in minutes)	Extraction Elapsed Time (%)
5	32	40.89	2.39	0.010	2.42
7	128	122.79	8.53	0.006	0.41
8	256	242.31	9.36	0.007	0.22

In Figure 8, we observe that the data-loading cost is negligible (in percentage) in comparison with the dataflow elapsed time and the invocation of the extractor program. The cost for executing the extractor program depends on the raw data file properties as well as the program chosen for extractions. As expected, the time of the extraction program increases as the size of raw data files also increases, since the external program has to manage more raw data. Therefore, the performance of the programs developed by users for extracting raw data can be directly associated to the extraction elapsed time, since it represents most of the time needed to access files and to gather raw data. We further highlight that these programs, as well as the amount of data to be extracted and registered along the dataflow, are chosen by the user. Depending on this cost, the user may choose to limit the dataflow queries and decide to analyze the raw data file contents isolated, outside Chiron. In this case, the user will still need to wait for this extractor program to execute to query the desired data element. Therefore, the time saved from not doing it with Chiron would be just the time of the data loading on the provenance database, which is negligible.



Extraction Program Execution	0.11 %	0.66 %	13.77 %	39.35 %
Data Loading on Provenance Database	0.34 %	0.61 %	0.89 %	1.05 %
Workflow Elapsed Time	99.89 %	99.34 %	86.23 %	60.65 %

Figure 8. Oil and Gas Extraction dataflow executions varying size of raw data files.

QUERY. Both dataflow case studies gather domain-specific data from raw data files. We now focus on the version of the Montage dataflow with extractors to analyze the potential of dataflow queries to debug, fine tune the dataflow execution and perform some domain-specific analyses. All queries were performed on relations that contain raw data extracted from files. We executed three SQL queries to represent each query type from Section 1, as shown in Figure 5.

The first query (Figure 9) allows for selecting domain-specific file content (type 1). In this case, we select the *CRVAL2* that corresponds to a *CRVAL1* attribute greater than 210. Its result is shown in Figure 10. These raw data values are extracted from a FITS file that was generated by a projection simulation program, and loaded into the *RProjection* table of the provenance database. This query will operate on *RProjection*, which represents a reduced view of the original raw data file. Alternatively, the result of this type 1 query can also be obtained without Chiron, by using a solution such as FastQuery [7], NoDB [9] and RAW [10]. However, our approach has several advantages. First, we allow for queries on selected FITS file attributes, through *RProjection*, while the simulation program is generating it. This feature may show to the user that the projection might not be going on the right direction or is producing anomalies, or even that needs to fine tune the configuration and the execution should be interrupted, the sooner the better. Another advantage is to avoid having to parse the FITS file after its generation to obtain the access path. In our approach, the selected attributes are gathered to the database while they were being written in the FITS file. Like the other approaches, we do not load the whole FITS file into our provenance database, just some selected attributes and pointers to files. These domain data stored in the provenance database may be seen as an index to the raw data file. Existing solutions do not support queries that consider file flow and data element flow.

```
SELECT CRVAL1, CRVAL2
FROM RProjection
WHERE CRVAL1 > 210.00;
```

Figure 9. Query for analyzing domain-specific file content.

CRVAL1	CRVAL2
246.401554	-27.1663622
246.5329709	-26.4132476
246.5329737	-26.682692
246.1365759	-27.1608872
246.5329764	-26.9521364
245.8708342	-27.1609012
...	...

Figure 10. Results for the first query.

The second query selects the produced files according to specific criteria (Figure 11). It selects the produced *HDU* files that have *CRVAL1* > 210.00 and *CRVAL2* < 60.00, considering simulation programs that present at least one projection result. Users may execute this query at runtime and check, as soon as possible, if these conditions occur for any data element in dataflow. It is important to note that this query requires a file flow that relates different raw data files, since FITS files need to be tracked in each step of the projection simulation. Furthermore, this query also supports the analysis of two extracted attributes (*CRVAL1* and *CRVAL2*) from raw data files, since we gather results according to these attributes. Therefore, this query reinforces the potential of our approach to track the manipulated files along the dataflow with a domain-specific criteria.

```
SELECT p.CRVAL1, p.CRVAL2, sp.HDU_AREA, sp.HDU_FILE
FROM RProjection p, RSelectProjections sp
WHERE sp.MOSAIC_ID = p.MOSAIC_ID
AND sp.ewkfid = p.ewkfid
AND sp.CNTR = p.CNTR
AND p.CRVAL1 > 210.00
AND p.CRVAL2 < 60.00
AND p.ewkfid = 1;
```

Figure 11. Query for tracing files based on domain-specific criteria.

As a third query, we specify and run a type 3 query to identify all linear transformations (*i.e.*, attributes *FA* and *FB* in relation *RFitPlane*) involved in projection simulation programs from Montage dataflow, considering specific mosaic input repositories (*i.e.*, attribute *REPOSITORY* in relation *RListFITS*). Based on this definition, this query analyzes the contents of linear transformations in custom mosaic generation, considering the data element flow between different data transformations (*i.e.*, *List FITS*, *Projection*, *Calculate Overlaps*, *Extract Differences*, *Calculate Difference* and *Fit Plane*). The tables involved in this dataflow query are presented in Figure 12. Therefore, this query traces back all the files along its derivation flow (*i.e.,* file flow management), while it also needs to consider the data element flow from different attributes among data transformations. For example, data element flow between data transformations *List FITS* and *Projection* is defined by attribute *MOSAIC_ID*, which is split in some tuples in relation *RProjection* for each FITS file in a mosaic repository (*i.e.*, Split Map operator). Meanwhile, data element flows is managed by attributes *MOSAIC_ID* and *CNTR* among data transformations *Projection* and *Calculate Overlaps*. Data element flow management is also ensured by attribute *MOSAIC_ID* to merge all projected FITS files (*i.e.,* attribute *TNAME*) and their linear transformations, and generate the custom mosaic in JPG file format (*i.e.*, attribute *MOSAIC_JPG*). Moreover, the third query may help users debugging dataflows at runtime, since it is possible to check on attributes from different data transformations (*e.g.,* attributes *FA, FB, CRVAL1* and *CRVAL2*) before or during the custom mosaic generation in JPG file format (*i.e.,* attribute *MOSAIC_JPG*).

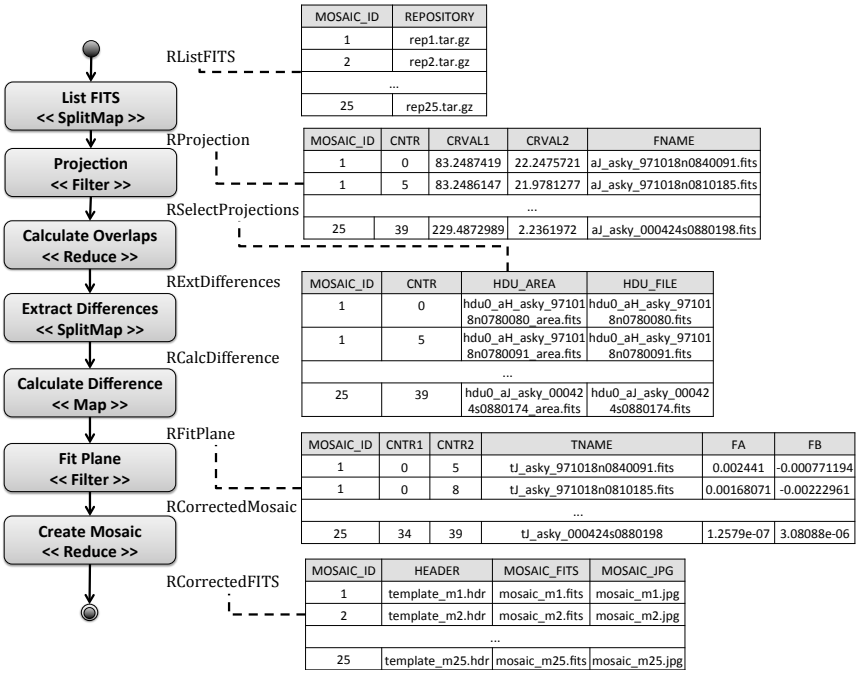


Figure 12. Data element flow for querying linear transformations in Montage dataflow.

6. CONCLUSION

Analyzing results in data-intensive computer simulations can be very hard. Users commonly have to analyze the domain-specific content from several thousands of files in order to confirm or refute a scientific hypothesis. At the same time, users also need to trace back the obtained results by relating the content of different files produced in the dataflow. This paper proposes an approach for raw data extraction in files consumed and produced in computer simulations. The proposed approach directly accesses domain-specific data stored in the produced data files represented in heterogeneous formats without having to first load all data into a DBMS. Just selected raw data is extracted and loaded into a provenance database to be further queried. The main advantage of the proposed approach is that element data is gathered at the same time it is being generated, instead of parsing the raw data file to allow for querying *a posteriori*. In addition to the domain-specific data gathering process, the workflow engine also registers the flow of transformations of the files and selected elements. Thus, our approach ensures file flow and data element flow management with support for complex domain data queries involving file and data element flow.

To evaluate the proposed approach, we used two dataflows as case studies: a Montage dataflow from astronomy and a synthetic dataflow based on Oil and Gas applications. One advantage of the proposed approach is that it benefits from the parallelism available in an HPC environment since several extractions can be performed concurrently. It speeds up the file processing by using more computing nodes when needed and the local cache. When executing the Montage dataflow, extracting and indexing the content of files through the provenance database yields an increase of 3.84% in total elapsed time. However, this cost for raw data extraction is paid off by the new capability of performing both file flow and data element flow analyses by querying data transformations. The Oil and Gas dataflow revealed a small cost for extraction process when it is data-intensive (*i.e.*, workloads with higher size of raw data files). In our experimental results for both dataflows, we noted a maximum cost for the extraction process of 3.7%. Overall, this experimental evaluation shows that important capabilities for raw data analysis can be provided at marginal cost through provenance management.

ACKNOWLEDGMENTS

Work partially funded by CNPq, CAPES, FAPERJ and Inria (MUSIC and HOSCAR projects) and performed (for P. Valduriez) in the context of the Computational Biology Institute (www.ibc-montpellier.fr). The experiments were executed at NACAD (COPPE / Federal University of Rio de Janeiro), currently an Intel Parallel Computing Center (IPCC). This research made use of Montage. It is funded by the National Science Foundation under Grant Number ACI-1440620, and was previously funded by the National Aeronautics and Space Administration's Earth Science Technology Office, Computation

Technologies Project, under Cooperative Agreement Number NCC5-626 between NASA and the California Institute of Technology.

REFERENCES

1. E. Pennisi, 2011, Will Computers Crash Genomics?, *Science*, v. 331, n. 6018 (Feb.), p. 666–668.
2. E.W. Greisen and M.R. Calabretta, 2002, Representations of world coordinates in FITS, *Astronomy and Astrophysics*, v. 395, n. 3 (Dec.), p. 1061–1075.
3. Unidata Program Center, 2014. NetCDF. URL: http://www.unidata.ucar.edu/software/netcdf/.
4. The HDF Group, 2014. HDF5. URL: http://www.hdfgroup.org/HDF5/.
5. J.C. Jacob, D.S. Katz, G.B. Berriman, J.C. Good, A.C. Laity, E. Deelman, C. Kesselman, G. Singh, M.-H. Su, et al., 2009, Montage: a grid portal and software toolkit for science-grade astronomical image mosaicking, *International Journal of Computational Science and Engineering (IJCSE)*, v. 4, n. 2, p. 73–87.
6. K. Wu, S. Ahern, E.W. Bethel, J. Chen, H. Childs, E. Cormier-Michel, C. Geddes, J. Gu, H. Hagen, et al., 2009, FastBit: interactively searching massive data, *Journal of Physics: Conference Series*, v. 180 (Jul.), p. 012053.
7. J. Chou, R.D. Ryne, M. Howison, B. Austin, K. Wu, J. Qiang, E.W. Bethel, A. Shoshani, O. Rübel, et al., 2011, Parallel index and query for large scale data analysis, In: *2011 International Conference for High Performance Computing, Networking, Storage and Analysis*, p. 1.
8. S. Blanas, K. Wu, S. Byna, B. Dong, and A. Shoshani, 2014, Parallel data analysis directly on scientific file formats, In: *2014 ACM SIGMOD International Conference on Management of Data*, p. 385–396.
9. I. Alagiannis, R. Borovica, M. Branco, S. Idreos, and A. Ailamaki, 2012, NoDB: efficient query execution on raw data files, p. 241.
10. M. Karpathiotakis, M. Branco, I. Alagiannis, and A. Ailamaki, 2014, Adaptive Query Processing on RAW Data, *VLDB Endowment*, v. 7, n. 12, p. 1119–1130.
11. K. Vahi, M. Rynge, G. Juve, R. Mayani, and E. Deelman, 2013, Rethinking Data Management for Big Data Scientific Workflows, In: *Workshop on Big Data and Science: Infrastructure and Services*, p. 27–35.
12. L. Assuncao and J.C. Cunha, 2014, Enabling Global Experiments with Interactive Reconfiguration and Steering by Multiple Users, *14th International Conference on Computational Science*, v. 29, p. 2137–2144.
13. S. Bowers, T. Mcphillips, S. Riddle, M.K. Anand, and B. Ludäscher, 2008, Kepler/pPOD: Scientific Workflow and Provenance Support for Assembling the Tree of Life, In: *2nd International Provenance and Annotation Workshop*, p. 70–77.
14. R. Ikeda, J. Cho, C. Fang, S. Salihoglu, S. Torikai, and J. Widom, 2012, Provenance-Based Debugging and Drill-Down in Data-Oriented Workflows, In: *IEEE 28th International Conference on Data Engineering (ICDE)*, p. 1249–1252.
15. E. Ogasawara, J. Dias, V. Silva, F. Chirigati, D. Oliveira, F. Porto, P. Valduriez, and M. Mattoso, 2013, Chiron: A Parallel Engine for Algebraic Scientific Workflows, *CCPE*, v. 25, n. 16, p. 2327–2341.
16. M. Mattoso, J. Dias, K.A.C.S. Ocaña, E. Ogasawara, F. Costa, F. Horta, V. Silva, and D. de Oliveira, 2014, Dynamic steering of HPC scientific workflows: A survey, *Future Generation Computer Systems* (Nov.)
17. J. Freire, D. Koop, E. Santos, and C.T. Silva, 2008, Provenance for Computational Tasks: A Survey, *Computing in Science and Engineering, v.10*, n. 3, p. 11–21.
18. V. Silva, D. Oliveira, and M. Mattoso, 2014, Exploratory analysis of raw data files through dataflows, In: *Workshop on Parallel and Distributed Computing for Big Data Applications (WPBA 2014)*, p. 114–119.
19. J. Kim, H. Abbasi, L. Chacon, C. Docan, S. Klasky, Q. Liu, N. Podhorszki, A. Shoshani, and K. Wu, 2011, Parallel in situ indexing for data-intensive computing, In: *IEEE Symposium on Large Data Analysis and Visualization (LDAV)*, p. 65–72.
20. B. Ma, A. Shoshani, A. Sim, K. Wu, Y. Byun, J. Hahm, and M.-S. Shin, 2012, Efficient Attribute-Based Data Access in Astronomy Analysis, In: *SC Companion: High Performance Computing, Networking, Storage and Analysis (SCC)*, p. 562–571.
21. Amazon EC2, 2010, *Amazon Elastic Compute Cloud (Amazon EC2)*, http://aws.amazon.com/ec2/.
22. Y. Gil, V. Ratnakar, J. Kim, P. Gonzalez-Calero, P. Groth, J. Moody, and E. Deelman, 2011, Wings: Intelligent Workflow-Based Design of Computational Experiments, *IEEE Intelligent Systems*, v. 26, n. 1 (Jan.), p. 62–72.
23. E. Ogasawara, J. Dias, D. Oliveira, F. Porto, P. Valduriez, and M. Mattoso, 2011, An Algebraic Approach for Data-Centric Scientific Workflows, *37th International Conference on Very Large Data Bases (PVLDB)*, v. 4, n. 12, p. 1328–1339.
24. R. Ikeda, A. Das Sarma, and J. Widom, 2013, Logical provenance in data-oriented workflows?, In: *2013 IEEE International Conference on Data Engineering (ICDE 2013)*, p. 877–888.
25. F. Costa, V. Silva, D. de Oliveira, K. Ocaña, E. Ogasawara, J. Dias, and M. Mattoso, 2013, Capturing and Querying Workflow Runtime Provenance with PROV: A Practical Approach, In: *Joint EDBT/ICDT 2013 - Workshops on EDBT'13*, p. 282–289.
26. P. Missier, K. Belhajjame, and J. Cheney, 2013, The W3C PROV family of specifications for modelling provenance metadata, In: *16th International Conference on Extending Database Technology*, p. 773–776.
27. R. Brun and F. Rademakers, 1997, ROOT — An object oriented data analysis framework, *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment*, v. 389, n. 1-2 (Apr.), p. 81–86.
28. E. Deelman, G. Singh, M. Livny, B. Berriman, and J. Good, 2008, The cost of doing science on the cloud: the Montage example, In: *SC '08: 2008 ACM/IEEE Conference on Supercomputing*, p. 1–12.
29. NASA/IPAC, 2014. Two Micron All Sky Survey (2MASS). URL: http://irsa.ipac.caltech.edu.
30. F. Chirigati, V. Silva, E. Ogasawara, D. Oliveira, J. Dias, F. Porto, P. Valduriez, and M. Mattoso, 2012, Evaluating Parameter Sweep Workflows in High Performance Computing, In: *1st International Workshop on Scalable Workflow Enactment Engines and Technologies (SWEET'12)*, p. 10.
31. NACAD, 2015. NACAD: High Performance Computing Center. URL: http://www.nacad.ufrj.br/. Accessed: 30 Jun 2015.