Raw data queries during data-intensive parallel workflow execution

Vítor J Silva, José J Leite, José J Camata, Daniel de Oliveira, Alvaro L G A Coutinho, Patrick Valduriez, Marta Mattoso

Raw Data Queries during Data-intensive Parallel Workflow Execution

Vítor Silva[1], José Leite[1], José J. Camata[2], Daniel de Oliveira[3],
Alvaro L.G.A. Coutinho[2], Patrick Valduriez[4], Marta Mattoso[1]

[1] Department of Computer Science, Federal University of Rio de Janeiro/COPPE, Brazil
[2] High Performance Computing Center and Department of Civil Engineering, Federal University of Rio de Janeiro/COPPE, Brazil
[3] Institute of Computing, Fluminense Federal University, Brazil
[4] Inria and LIRMM, France

{silva,jvdleite,marta}@cos.ufrj.br; {camata,alvaro}@nacad.ufrj.br; danielcmo@ic.uff.br;
patrick.valduriez@inria.fr

Abstract

Computer simulations consume and produce huge amounts of raw data files presented in different formats, *e.g.*, HDF5 in computational fluid dynamics simulations. Users often need to analyze domain-specific data based on related data elements from multiple files during the execution of computer simulations. In a raw data analysis, one should identify regions of interest in the data space and retrieve the content of specific related raw data files. Existing solutions, such as FastBit and RAW, are limited to a single raw data file analysis and can only be used after the execution of computer simulations. Scientific Workflow Management Systems (SWMS) can manage the dataflow of computer simulations and register related raw data files at a provenance database. This paper aims to combine the advantages of a dataflow-aware SWMS and the raw data file analysis techniques to allow for queries on raw data file elements that are related, but reside in separate files. We propose a component-based architecture, named as ARMFUL (Analysis of Raw data from Multiple Files) with raw data extraction and indexing techniques, which allows for a direct access to specific elements or regions of raw data space. ARMFUL innovates by using a SWMS provenance database to add a dataflow access path to raw data files. ARMFUL facilitates the invocation of *ad-hoc* programs and third party tools (*e.g.*, FastBit tool) for raw data analyses. In our experiments, a real parallel computational fluid dynamics is executed, exploring different alternatives of raw data extraction, indexing and analysis.

Keywords

Scientific workflows; Dataflow; Raw data analysis; Index raw data.

1. Introduction

Several data-intensive computer simulations take a long time to execute even when using High Performance Computing (HPC) environments. When these simulations dataflows are managed by a parallel Scientific Workflow Management System (SWMS) [1], they benefit from provenance [2] and data parallelism among different programs that compose the workflow. Systems like Swift/T [3] and Pegasus [4] are highly scalable SWMS and have shown impressive performance results for many different scientific application domains [3,5].

A data analysis challenging problem occurs when users have to navigate and browse thousands of raw data files that result from these data-intensive simulations. Provenance data from SWMS are an important asset in relating these files, but still very far from supporting raw data analytical queries in these file contents. There are several solutions to improve data analysis through query processing on raw data file contents [6–9]. They typically parse the raw data file; extract relevant contents; index; and present query support, usually with the help of a Database Management System (DBMS). However, these solutions do not support queries that correlate elements from different raw data files.

Combining provenance support from SWMS and raw data query solutions bring a new vision to scientific data analysis. Current solutions are independent and offline [10]. This means that users are only able to query provenance or raw data when the execution finishes. In data-intensive computer simulations, workflow executions may take very long to execute (hours or days) even in HPC environments [5]. Typically, a user tries several different workflow configurations before reaching satisfactory parameters, convergence and error values. This

requires a runtime data analysis support, where users may abort or fine tune, and debug their workflow long before it finishes.

In a previous work [11] we have shown the advantages of adapting loop conditions based on partial data analysis, all during the iterative workflow execution. We used Chiron SWMS [12] and its data-centric algebra [13] to query provenance data related to domain data using a specialization of W3C PROV [14]. The work in [11] evolved into defining workflow algebraic operations to control and adapt loop conditions [15]. In a study on uncertainty quantification we anticipated a convergent state and dynamically changed loop control [15]. Such actions contributed to reduce execution time in several hours. These simulations or other iterative workflows typically take several hours or days to execute. These results led us to improve raw data analysis support into selecting raw data elements from files and relating them to workflow parameters at runtime [16]. However, our previous solutions suffer from limited raw data access with no direct path to specific regions or elements of raw data files. Users still had to write specific programs to access and analyze the raw data files. Even though direct access to files can be obtained by querying the provenance database, the raw data file content analysis remains isolated from the provenance database.

There are several challenges in providing raw data file analysis, while the workflow is being executed by the SWMS. Performance is critical; the overhead in runtime data analysis support must not harm the parallel workflow execution time. Another challenge is managing the large size of these files with their specific raw data format; they cannot be converted to be inserted (*i.e.*, replicated) into a database for queries. There is also the issue of mapping and accessing specific regions of interest inside the raw data file.

To address these challenges and benefit from SWMS provenance data with raw data analysis support, the following services must be provided during the workflow execution:

- access to raw data files while they are being generated;
- parse raw data to find relevant data;
- extract relevant subsets of raw data;
- index over data regions of interest;
- prepare raw data for queries;
- runtime query relating raw data from different files, provenance data, and performance execution data.

In this paper, we present a raw data analysis support to address these challenges. We implemented all these services in an architecture named ARMFUL (Analysis of Raw data from Multiple Files) that can be plugged in SWMS. In its current version, it is implemented in an extended version of Chiron and evaluated in an HPC environment with a real numerical simulation workflow. Experiments with a finite element solver for fluid dynamics [17] workflow show the performance improvements in runtime queries obtained by the ARMFUL indexing techniques. The results present relevant costs/benefits considering the overheads of managing and indexing raw data, with further gains from powerful runtime data analyses obtained by queries accessing data directly.

The rest of the paper is organized as follows. Section 2 describes a motivating scenario from numerical simulations. Section 3 discusses related work on raw data analysis from the execution of computer simulations. Section 4 defines dataflow concepts with workflow algebra operations to analyze raw data files. Section 5 presents ARMFUL, an architecture to support raw data analysis, which is based on raw data extraction and index generation. Section 5 also presents how we use Chiron SWMS to implement ARMFUL. Section 6 uses the motivating example from Section 2 to present the experimental results. ARMFUL raw data support is evaluated with real computational fluid dynamics workflows, while extracting, indexing and querying raw data from XDMF and HDF5 files. Finally, Section 7 concludes.

2. Motivating Scenario

To clarify the exploratory analysis of raw data files, let us consider an example from Computational Fluid Dynamics (CFD) that is presented in Figure 1. Specifically, this simulation analyzes the three-dimensional flow of an incompressible fluid in a cavity. This problem is a popular benchmark in CFD, used to evaluate new codes or new solution methods [18]. This benchmark is consistently used throughout this paper. In Figure 1, the black boxes show the simulation programs invoked for the configuration of the CFD analysis (*edgecfdPre* program) and solver execution (*edgecfdSolver* program). The *edgecfdPre* program generates configuration files in *part.in* and *part.mat* formats, which contain the needed raw data elements, such as the maximum time step (attribute *DTMAX*), the final simulation time (attribute *TMAX*), material properties such as fluid viscosity (attribute *VISC*) and fluid density (attribute *DENS*). The *edgecfdSolver* program consumes the produced files by *edgecfdPre* program, and generates solution attributes (such as velocity and pressure) over the execution of the CFD solver (*i.e.*, files in XDMF and HDF5 formats) and some metrics for analyzing the CFD model convergence (number of linear iterations and

residual norms represented as attributes *ITER* and *RESIDUALS*, in STP file format). The attribute *RESIDUALS* represents the ratio of the Euclidean norm of the momentum and continuity residuals between two linear iterations.

Users in CFD domain commonly investigate the convergence of their solver based on the number of linear iterations and residual norms for a specific input mesh, boundary, initial conditions and material properties. To access solution quality and/or to define if it is necessary more time steps, users may increase the value of *TMAX*. This means that the CFD solver needs more time to reach a steady state, and other controlling parameters may also be adjusted. To analyze the number of linear iterations and residual norms, users must gather raw data elements (*e.g.*, values for *ITER* and *RESIDUALS*) from STP files. To narrow their analysis for a specific input mesh and fluid properties and correlate it to the solver properties, users have to investigate the dataflow path for a specific input mesh (*e.g.*, *cav.1.msh*, *cav.1.part.in*, *cav.1.xmf*, and *cav.1.stp* files). Furthermore, checking if the solution has reached the steady state (solution is no longer changing) is based on the reduction on the number of linear iterations and residual norms over time, as presented by the behavior in charts of scenario (a) in Figure 1. A solver execution that is not able to converge is represented by the behavior of scenario (b) in Figure 1. The analysis of scenario (b) can assist users to abort CFD executions that will not converge or to change some attributes like *TMAX*. The modifications on *TMAX* need a deep analysis in several parameters and raw data file contents to assert that the computational model is able to converge in more time steps.

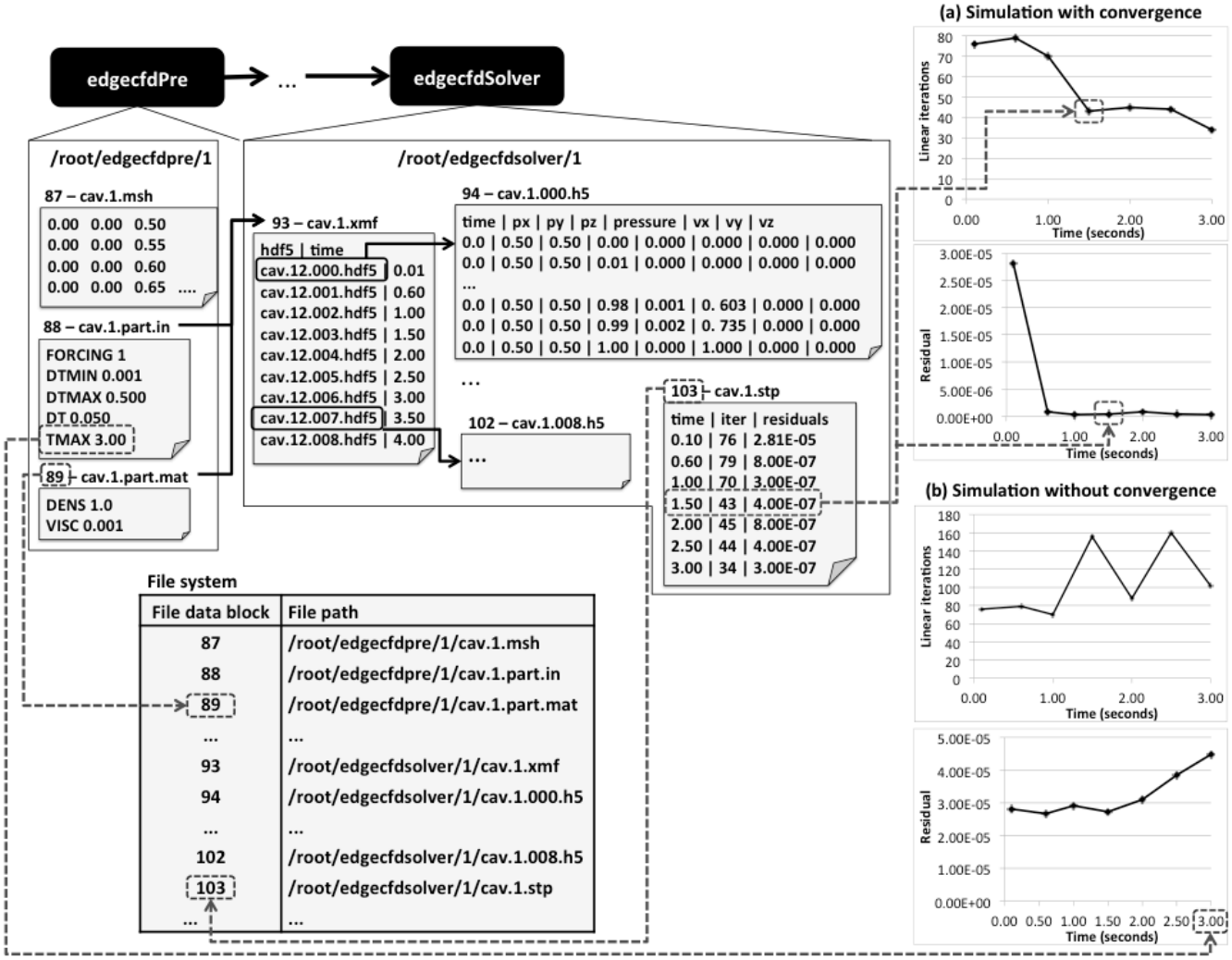


Figure 1. Raw data analysis in CFD simulation.

To support the given examples of raw data analysis, a system underlying the dataflow generation of pre, post-processing and the solver itself would have to trace and register simulation data for queries at runtime. Let us consider a scenario with no dataflow analysis support. After finding the directories and relationship between files *cav.1.msh* and *cav1.xmf*, the user would need an external program to find which elements from the first file relates to which elements of the latter. Then, another external program would have to be used or programmed to browse *cav1.xmf* to find the element *cav.1.008.h5* inside the file and continue from there, all manually. In addition, changing a loop condition (*e.g., TMAX*) requires an external loop execution control, to be able to dynamically change it, and react to these changes, like keeping consistency in dataflow generation, as we show in [15].

However, the decision of when or how to change requires deeper raw data analysis support, currently not found in any work, to the best of our knowledge. This situation can be much worst if the user has to run and monitor many large simulations at the same time, *e.g.*, when varying material parameters (parameter sweep) or dealing with the propagation of uncertainties in initial conditions or material properties (uncertainty quantification [15]).

3. Related Work

In this section, we discuss the existing solutions for raw data analysis of files generated by workflow execution. We consider the support for the six services described at the Introduction: access; parse; extract; index; queries; and runtime query relating raw data from different files, provenance data and performance execution data.

Current solutions do not provide for all six services and fall basically in two categories. The first category considers raw data file analyzers [7–9,19–22], which support the services of: access; parse; extract; index; and sophisticated queries on raw data file contents. However, this raw data support is offline (analyzes only after the execution of the simulations) and consequently falls short in relating different raw data files or any provenance database. The second category corresponds to SWMS with provenance data support, like [3,4,12,23–26]. Through provenance data, raw data files are identified as input and output of workflow activities. However, queries are limited to relating files with workflow provenance data, but no raw data file content analysis is supported, and they are all offline analyses. So, raw data analysis techniques would be needed to browse the file contents and access data elements from the files.

Unlike Chiron [12], current SWMS do not allow for provenance data analysis at runtime. Our previous SWMS solutions are an exception in this scenario of raw data analysis at runtime. In [13], we present a solution to allow for provenance queries at runtime. In [11,27], we show how parameters such as loop control can be queried, steered and changed at runtime. In [16], we present an *ad-hoc* dataflow raw data support to complement provenance based queries. In this work, we build on these previous solutions by formalizing a dataflow approach considering data element flows with indexing techniques, all made available for raw data analysis with provenance at runtime even in HPC environments.

3.1. Raw Data Extraction from Files

Users commonly need to extract raw data from files [16] for performing domain-specific analysis. This process, known as raw data extraction, may present two granularity levels according to the volume of data retrieved from files: total and partial extraction. Total raw data extraction retrieves all attribute values from a raw data file, *e.g.*, XDMF feature for ParaView[1] toolkit. Partial raw data extraction retrieves part (or slices) of the attribute values from a raw data file, *e.g.*, HDF5 tools [22] and SDS framework [20]. Moreover, there are related work that extract raw data from files on demand, *i.e.*, attributes are extracted from files as requested by users (in some specific time steps). This approach is named as incremental raw data extraction. Also, a raw data extraction commonly needs to tokenize and parse retrieved data [28], since raw data files often contain binary or semi-structured data that have to assume data structures acceptable by the DBMS. Although raw data are extracted, tokenized, and parsed, there are some circumstances that users may need to generate indexes to provide direct access to the raw data in a specific region of interest from files.

3.2. Indexing Raw Data from Files

The index generation improves direct access to specific regions of the raw data space. The definition of the most suitable indexing algorithm depends mainly on: (i) the volume of scientific data manipulated, (ii) the data structure that is intended to be captured and (iii) the analysis to be performed on the computer simulation at hand [8,28]. Since workflows with only raw data extraction may present a severe overhead for data ingestion into the external repository (*e.g.*, file or database) in a large-scale computer simulation, it fits in cases that the computational model generates a small volume of data on each data transformation or a few attributes are extracted from each raw data file and those attributes assume standard data structures (*e.g.*, floating-point, integer, and characters sequence).

Otherwise, index generation can be employed when there is a large volume of data to be extracted and the data structures are complex, such as octrees and meshes [7,28]. In this paper, we present file indexes to address the raw data elements with more complex data structures. Indexing algorithms can reduce the time to find which subset of raw data is relevant for the analysis. Although index generation offers advantages especially to query processing, it has its own overheads, which have to be considered when deciding what to index [8,9].

[1] http://www.paraview.org/

There are different existing solutions in scientific domains that already use raw data indexing algorithms, such as FastBit tool [9] with bitmap indexes, and DiNoDB [19] with positional indexes. Indexing algorithms based on bitmaps, known as bitmap indexing, consist on the domain analysis of certain attributes in the raw data files to generate Boolean indexes through the verification of algebraic equations. For example, assuming that the values of attribute *X* for all scientific data files present a domain with only four possible values in the integer data structure, then the bitmap needs four columns to evaluate the presence of certain value, admitting an equality equation. In some cases, the use of inequalities for a bitmap indexing may be more beneficial. Supposing a floating-point number with a very extensive domain, the use of inequalities can be more suitable to generate indexes (less number of bitmap columns) and query certain intervals (of values) of the attribute indexed (search space limited by the generated indexes). FastBit tool, FastQuery [29] and SDS/Q framework are examples of related work that employ bitmap indexing. However, none of these solutions can manage data element through dataflow generation.

Different from bitmap indexing, positional indexing makes use of information that facilitates the raw data location (position) in files. A positional index can be two integers (or other standard data structure), where the values determinate where to start to read an attribute in the raw data file and the length in bytes of the attribute value. Positional indexing provides a smaller data overload to manage scientific data, since it only generates a pointer (two integer values, for instance) for each attribute value, while bitmap indexing presents a bitmap for each attribute with redundant bits (zero sequences). Another difference between both indexing algorithms is the ability that positional indexes have to reference complex data structures, like trees and multidimensional arrays. NoDB [7], DiNoDB, and RAW [8] allow for positional indexing of raw data from files.

3.3. Raw Data and Workflows

Raw data exploratory analysis is not limited to the investigation of scientific data in a single file. In many cases, users analyze the dataflow path to scan data propagation over multiple related raw data files. Therefore, the approaches for raw data analysis must be able to monitor the flow of the data consumed and produced by each simulation program during the execution. As presented in [16], there are basically two abstraction levels to deal with dataflow management: physical and logical.

Dataflow management at the physical level consists in supporting data transformation on the file system. Thus, this level of dataflow management deals with files as black boxes, since there are no indexes or query support to access their domain-specific content. Hence, users are limited to perform analysis with pointers to the related files in the dataflow, which creates the need to analyze each file individually or develop specific programs to extract and index their domain-specific content. Both alternatives present tedious and error-prone tasks.

In contrast, dataflow management at the logical level deals with the monitoring of how data elements are consumed and produced by the simulation programs. Those data elements can be the raw data from files or the data propagated through the chained simulation programs. The relationship between data elements may be used to trace the dataflow path, which contributes to the analytical potential of this management level. From the users' point of view, it is possible to query domain data related to the execution of the computer simulation without the need to develop specific programs to extract raw data from multiple files. On the other hand, the elapsed time of the computer simulation is higher for dataflow management at the logical level, since the data elements must be extracted and monitored at runtime, while at the physical level such monitoring comes down to pointers to the raw data files. Considering the dataflow management on both levels, users need to capture and store provenance data [2] about the composition of the computer simulation (and their data dependencies) and raw data gathered from files in each simulation step (*i.e.*, invocation of a simulation program).

As shown in [16], the AWARD framework [26] has a dataflow approach based on the physical level. It captures tuples (set of parameter values) at runtime and manages data propagation through simulation programs according to the dependency between existing data. Despite treating the representation of parameter values, AWARD does not handle parameter values as data elements in its relational representation, which prevents the management of the dataflow at the logical level. There are some SWMS with dataflow management support at the logical level, such as Kepler [24], Panda [25] and Chiron [12]. SWMS typically register dataflow as workflow provenance. However the support in Kepler and Panda for provenance data analysis is offline, *i.e.* they do not provide any runtime data analysis feature. All provenance data analyses have to be performed before or after the workflow execution. In addition, although Kepler can support dataflow management at a logical level, it does not go into file contents. With respect to Panda, it supports a provenance data formalism that represents dataflow produced by a workflow composed of relational algebra operations. However, Panda aims at textual structured data from databases, not facing the problem of raw data files.

Chiron uses a workflow algebra to model and execute workflows. This algebra is inspired by the relational algebra for databases [30] and provides a uniform data model that expresses all workflow data as relations. It is a parallel SWMS, where workflows are represented and executed as algebraic expressions composed of operations

that encapsulate simulation programs and operands to represent the dataflow. Different from related work, Chiron enables capturing and storing provenance data ready to be queried at runtime. It allows raw data file analysis through external program invocations, which extract domain-specific content from raw data files. In this paper, we generalize and extend Chiron's raw data support by isolating the components that manage raw data from the SWMS, making them available as services of the ARMFUL architecture. While related work does not consider file contents in runtime analyses, we use the dataflow abstraction to go "inside" the files. In addition to associating one input file to a corresponding output file, we allow for associating one element from an input file to its corresponding output element in its output file, as the workflow executes. Therefore, the input dataset concept is not just a set of files, but also a set of elements within a raw data file.

Comparing these related work with our contributions in the present paper, we adapt Panda's formalism for defining dataflows with provenance data from workflows to be used as operands for the algebraic operations we propose here for managing raw data. To the best of authors' knowledge, Chiron is the only SWMS supporting raw data file analysis, since it extracts raw data from files and ingests them in a provenance database. However, considering large-scale computer simulations, Chiron presents a significant overhead when a large amount of raw data is ingested into its relational DBMS. Also, this is the first work that allows dataflow management at the logical level considering raw data extraction and index generation combined with provenance data at runtime.

4. Using a Dataflow Approach with Workflow Algebra to Analyze Raw Data Files

Workflow algebras are focused on the operations and not the datasets that flow from one data transformation to the other. To represent, manage and access the datasets of the workflows, they have to be explicitly represented as a data transformation of a dataflow notation. In this section, we present dataflow concepts related to a workflow algebra. To consider raw data file analysis, we introduce two algebraic operators to represent the raw data extraction and indexing from files generated by scientific programs (*i.e.*, data transformations).

4.1. Dataflow Definition

We provide formal definitions for dataflow concepts to be used in the remaining of this paper. We define that each data set (Definition 1) is any set of data elements, which has some predefined attributes in each data element. In a dataflow scenario, a data transformation (Definition 2) consumes data from one or more data sets (inputs) and produces one or more data sets (outputs). The chaining of these data transformations and the specification of the involved data sets represent the dataflow (Definition 3).

Definition 1. (Data set) Data set S *has* a set of data elements $E = \{e_1,e_2,...,e_x\}$, where x is the number of data elements. Each data element has a sequence of predefined attributes $A = \{a_1,a_2,...,a_y\}$, where y is the number of attributes. Thus, each data element has values for each of these predefined attributes.

Definition 2. (Data transformation) Data transformation T is characterized by the consumption of one or more input data sets I and the production of one or more output data sets O. Formally, a data transformation is represented by T, where $O = T(I)$.

Definition 3. (Dataflow) 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 O. Composition is associative, so the linear composition of n data transformations is denoted as $T_1 \circ T_2 \circ ... \circ T_n$. Such composition of n data transformations is denoted 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 O from this dataflow can be represented as $O = (T_1 \circ T_2 \circ ... \circ T_n)(I_1)$, or $O = (D_F)(I_1)$ for short, where I_1 represents the input data sets of this dataflow. This definition is inspired on the concepts presented by Ikeda *et al.* [31].

4.2. Workflow Algebra meets Dataflow Concepts

While a dataflow notation represents the data sets of transformations, the operations behind these transformations are defined in data-centric algebraic workflow notations. The two notations are complementary. There are several workflow algebras based on data set transformations (*i.e.,* workflow activities) [13,32,33,23]. These transformations are ruled by algebraic operators (*e.g.,* Map, Reduce, Filter, and Flat Map). Those algebraic operators are data-oriented. Operands of the algebraic operators represent the data sets of the dataflow. In particular, the workflow algebra proposed by Ogasawara *et al.* [13] is capable of modeling entities as datasets and its relationships. The execution control is based on data dependency synchronization, but the algebra also has loop control and branch control operations [15]. For instance, if a data transformation T_i generates I_{i+1} that is a subset of I_i, we may say that this T_i behaves like a Filter operator. When T_i operates on I_i, this means invoking a function or a program represented by T_i with an operand I_i. The algebraic operator just rules the behavior of T_i with respect to the data set transformation. In SWMS, a data transformation corresponds to a workflow activity.

The dataflow $O = (T_1 \circ T_2)(I_1)$ can be represented by the following algebraic expression:

$$I_2 \leftarrow \text{Operator}_1\ (T_1, \text{optional parameters}, I_1)$$
$$O \leftarrow \text{Operator}_2\ (T_2, \text{optional parameters}, I_2)$$

This expression notation combines the explicit representation of operators (from workflows) and operands (from dataflows), such as input, intermediate and output data sets. Considering that in the previous expression, Operator_1 is a Map, its execution applies T_1 to all elements of I_1, generating for each input element of I_1, a corresponding output element in I_2. However, despite the access to elements of the data set, in workflow algebras, elements are not explicitly represented. Since we use a relational algebra based approach to represent dataflows with workflows, each T_i of a D_F is associated to a workflow algebraic operator and its operand I_i and result I_{i+1} are associated to relations. The set of attributes A, from data elements E, from I (Definition 1), correspond to attributes of tuples in these relations, respectively. Having an explicit representation for elements in I from a D_F allows for establishing an element flow, which has shown to be an important asset in workflow data analysis [16]. For textual data sets, loading data elements into relations is trivial, but when the elements of data sets are raw data files, it requires specific operations to parse the file to extract data element values from these files. When a raw data file is seen as a "black box" set in a workflow execution, data elements implicitly related from multiple raw data files are not addressable for queries.

4.3. An Algebraic Operator for Raw Data Extraction

In this subsection, we present an algebraic operator to extract raw data elements from files in dataflows. It aims to be consistent with data-centric workflow algebra operators. The operator Raw data extraction (Raw, for short) enables to extract selected data elements from raw data files to insert into elements of workflow data sets. The Raw algebraic operator encompasses raw data parsing of input data sets, selection of specific values for extraction and loading these values into elements of the output data sets. The Raw operator is represented as:

$$I_{i+1} \leftarrow \text{Raw}\ (T_i, RC, RF, I_i)$$

where the execution of Raw applies T_i to all elements of I_i, generating for each input element of I_i, corresponding output elements in I_{i+1}. T_i represents a workflow activity that invokes a program that is able to access, parse and extract raw data from the files referenced in I_i. To apply T_i as a Raw data extraction operation on raw data files, it requires two additional parameters: *RF*, which has the name of the I_i attribute that has, in each tuple, a raw data file name, and *RC*, which is a set of the identifiers of raw data file contents to be extracted from each raw data file and stored in a corresponding output attribute of I_{i+1}.

Let us consider a fragment of the motivation example presented in Section 2. The following algebraic expression defines a Map operator that invokes the *edgecfdSolver* program, represented as T_1 followed by a Raw operator, represented as T_2, which allows for raw data analysis on S, the output of T_1 (the *edgecfdSolver* program):

$$S \leftarrow \text{Map}\ (T_1, R)$$
$$V \leftarrow \text{Raw}\ (T_2, \{TIME, POINT\_0, POINT\_1, POINT\_2, VELOCITY\_0\}, HDF5, S)$$

Figure 2 illustrates the operand relations *R, S* and *V* with tuples that are generated as the workflow is executed. The boxes in Figure 2 represent program invocations responsible for data transformations and <<*stereotypes*>> represent the algebraic operator that rules the data transformation. Since the Map operator rules T_1, each activation of T_1 is invoked consuming a tuple from its input relation *R*. The *R* attributes are defined as follows, *ID:* the primary key; *MESH*: a binary input file, which represents a mesh for CFD simulations; *VISC*: a numerical attribute value to represent the fluid viscosity; and *DENS*: a numerical attribute to be used by T_1 as the fluid density. Each of the *n* activations of T_1 consumes one tuple from *R*, and produces one corresponding tuple in *S* (this is the typical behavior of a Map operator), where its attribute *HDF5* represents the HDF5 file path that is produced by T_1.

The values for attributes *VISC* and *DENS* in *R* represent a parameter sweep [34] scenario, where users want to run their CFD solver for different material properties (different attribute values to the fluid viscosity and density). The Raw operator is expressed so that the raw data files from the attribute *HDF5* of *S* are parsed (as specified by the parameter *RF*) to extract values for the raw data contents of *TIME*, *POINT_0*, *POINT_1*, *POINT_2*, and *VELOCITY_0* (as specified by the parameter *RC*) in its output relation *V*.

As *V* is generated as a database relation, users may submit queries to *V* and relate its elements to tuples of *S* and *R* at runtime. Users may, for example, select fluid simulations with a specific range for the average velocity of a fluid in *x* coordinate (*VELOCITY_0* output values) in a specific time (attribute *TIME* in *V*). Moreover, this component may also correlate *ID* output value (from *V*) with the input mesh (attribute *MESH* in *R*) consumed by the CFD solver. Therefore, the Raw operator is able to enrich the operand relations with domain-specific data from raw data files. These domain-specific data are related along the workflow execution allowing analyzing raw data as dataflows.

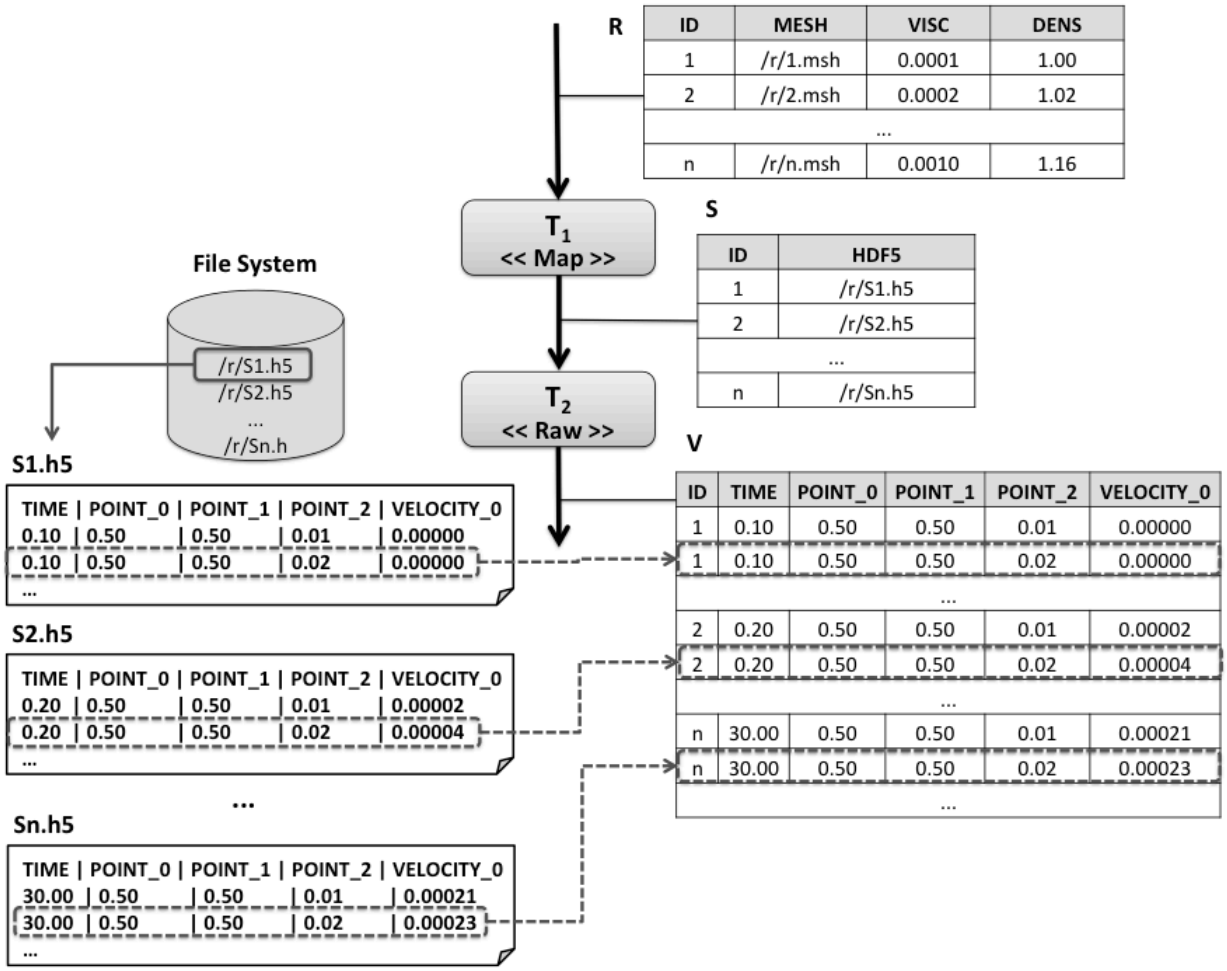


Figure 2. Data transformations with the Map and Raw operators.

4.4. An Algebraic Operator for Raw Data Indexing

We propose a new algebraic operator for indexing elements from raw data files in dataflows, also integrated to the workflow algebra operators. The operator is named Raw data Indexing (RawI, for short), as follows:

$$I_{i+1} \leftarrow \text{RawI}\ (T_i, RC, RF, I_i)$$

where, similarly to the Raw operator, the execution of RawI applies T_i to all elements of I_i, generating for each input element of I_i, corresponding output elements in I_{i+1}. T_i represents a workflow activity that invokes a program that is able to access, parse and index raw data from the files referenced in I_i. To apply T_i as a RawI data indexing operation on raw data files, it requires two additional parameters: *RF*, which has the name of the I_i attribute that has, in each tuple, a raw data file name, and *RC*, which is a set of identifiers in the raw data file contents to be indexed and stored in a corresponding output attribute of I_{i+1}.

RawI accesses raw data elements from files and employs indexing techniques to provide direct access to specific regions of the raw data space in files. Since there are well known solutions for indexing raw data from single files, such as FastBit, NoDB, and RAW (as discussed in Section 3.2), the invocation of these indexing implementations can be specified in the algebraic expression. The idea is that some pre-built indexing algorithms can be made available to be used as data transformations in the specification of RawI.

Let us consider the same example of the Raw operator (Figure 2), replaced here by the RawI operator as in the following algebraic expression:

$$S \leftarrow \text{Map}\ (T_1, R)$$

$$V \leftarrow \text{RawI}\ (T_2, \{TIME, POINT\_0, POINT\_1, POINT\_2, VELOCITY\_0\}, HDF5, S)$$

Figure 3 illustrates an example of a dataflow that presents a data transformation for indexing raw data, represented as T_2 and ruled by RawI operator. The indexing program invoked by RawI accesses the raw data elements related to the contents specified by the argument *RC* from files (in argument *RF*) and generates indexes to provide direct access to the data elements.

The indexing program T_2 generates index files to store all generated pointers to the raw data contents in a file. In this case, T_2 invokes the FastBit tool and generates a bitmap index for the raw data contents, which is stored in an index file with the IDX file extension.

The output relation of RawI plays the role of a metadata catalog. The query processor is able to identify that the *V* attributes *TIME* and *VELOCITY_0* are indexed and uses the corresponding index files (/1/TIME.idx and /1/VELOCITY_0.idx in Figure 3) to directly access the file contents. The FastBit tool has a program named *ibis* that is invoked when queries are submitted to *V*. For example, if the user wants to access regions that contain velocity values > 0.8, all VELOCITY_0.idx index files are evaluated by the FastBit tool that directly accesses the selected regions. The advantages of using RawI in comparison with Raw are that the same queries can be executed with less raw data in its database. A single index file represents the values of several raw data contents and the elapsed time needed to generate an index is smaller than the needed time to extract raw data.

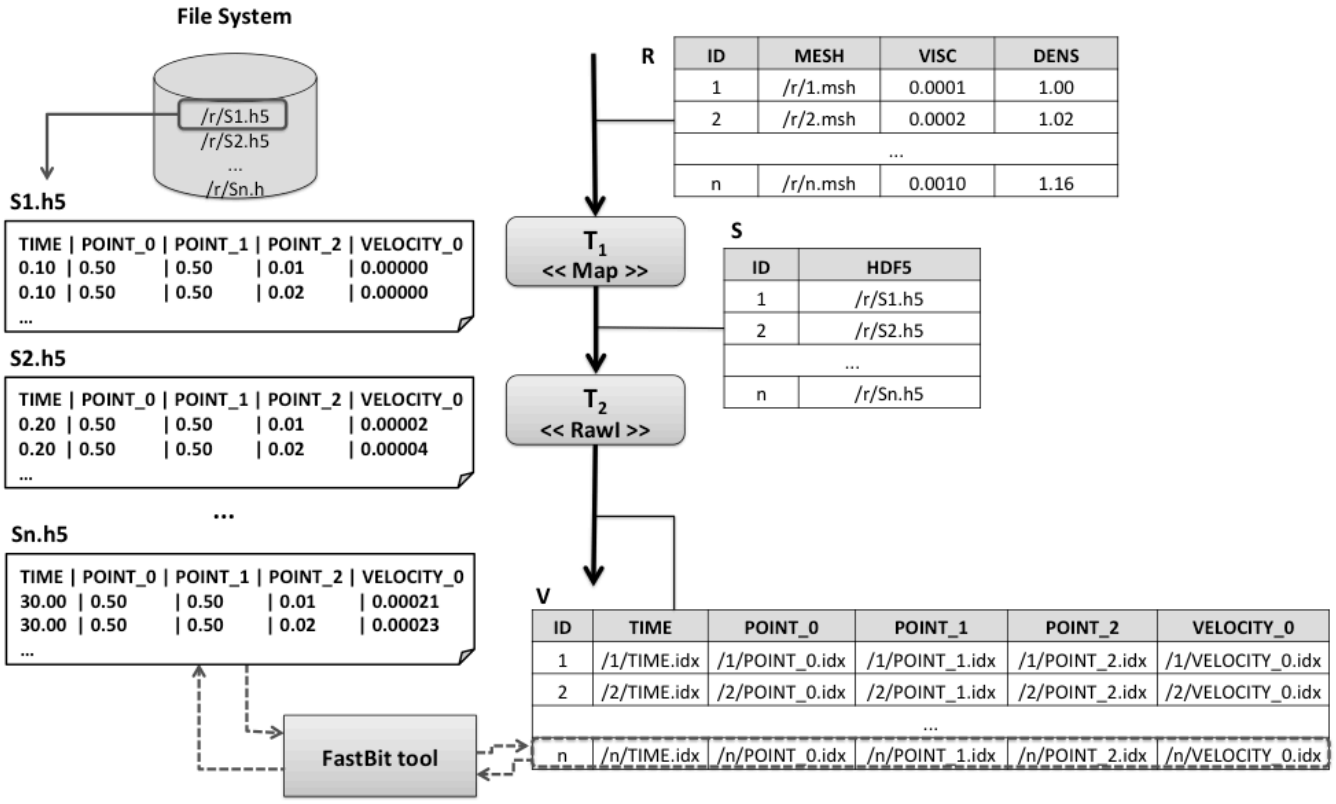


Figure 3. Data transformations and indexing with the Map and RawI operator.

5. ARMFUL: An Architecture to Analyze Raw Data from Dataflows

This section presents ARMFUL, a component-based architecture that enables the Analysis of Raw data from Multiple Files. ARMFUL analyzes scientific data by retrieving raw data from files and relating them through dataflow provenance data. Raw data management requires features that are domain-specific, therefore, provenance data models leave this representation in a coarse grain level. However, provenance queries have limited analytical value if not related to domain-specific data elements. This requires a lot of effort from SWMS users in developing, for each domain, data modeling design and specific programs to access, extract and relate domain data to provenance data. The ARMFUL architecture aims at helping on this effort by presenting some generic components that model and relate domain-specific data to provenance in the same database.

Since the raw data file contains a data set (in a binary or other encoding), the extraction process needs to identify elements within this data set. As the identification of data elements requires data access that depends on the application domain, we isolate those dependencies in cartridges [35]. A cartridge encapsulates a method that is able to access the contents of a raw data file and returns the data elements. This method is typically implemented by a program known by the user (domain specialist), who has to define it during the workflow specification. Therefore, the goal is to leave the domain data specificities to the cartridges that are invoked by ARMFUL components.

We chose a relational DBMS to manage provenance and raw data to benefit from its analytical query support in dataflows. DBMS present well-known algorithms and strategies to guarantee atomicity, consistency, isolation, and durability in transactions, while they also have consolidated solutions for enabling concurrency control and recovery [36]. Therefore, ARMFUL extracts selected content from raw data files, generates indexes (as an optional

step), and allows for using it to query raw data combined to provenance data through the DBMS. In addition, ARMFUL has to manage the relationships among files, needing file references to access and extract raw data from files. These characteristics are essential for enabling dataflow management at the logical level. They enable queries that associate raw data elements from multiple files, *i.e.*, from different data transformations (execution of simulation programs), as discussed in the analysis of Figure 1. Following, we discuss each one of the ARMFUL components in detail.

Figure 4 presents the components of ARMFUL. As in SWMS with provenance support, white components correspond to the capture of provenance data and their storage in a provenance database. Gray components (*Raw Data Extraction*, *Raw Data Indexing*, and *Query Processing*) describe the steps to extract raw data from files, to generate indexes over the raw data, and to query provenance and raw data from the same database. These three components are detailed in the following subsections. The definition of the most suitable strategy to be used (raw data extraction and index generation) depends on the volume of data and the data structures of the specific contents in raw data files, as discussed in Sections 3.1 and 3.2.

The *Provenance Data Gathering* component aims at capturing provenance data and execution metadata generated during the parallel execution of scientific workflows, and storing those gathered data in a provenance relational database. To enhance raw data analysis support, this component also manages the file references and the data dependencies (relationships) among raw data files. File references are essential to raw data extraction and indexing as aforementioned, while data dependencies enable dataflow management at the logical level. Therefore, this component combines provenance with some metadata about the raw data to be extracted/indexed by *Raw Data Extraction* and *Raw Data Indexing* components.

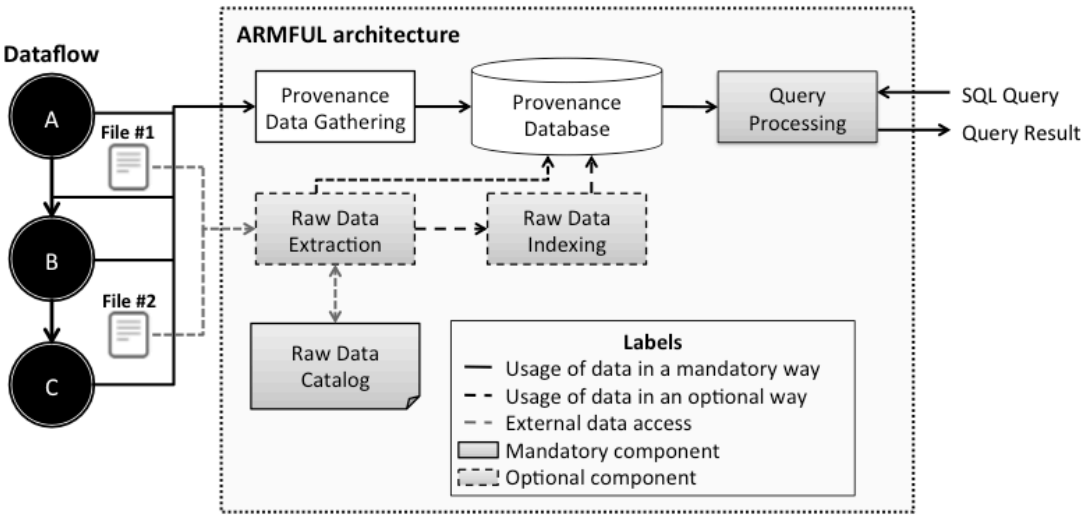


Figure 4. Components of ARMFUL.

5.1. Raw Data Extraction and Indexing Components

Raw Data Extraction component aims at reading the contents of raw data files, parsing them and retrieving selected contents that are relevant according to the attributes chosen by the user. To achieve this goal, this component follows four steps: content read, tokenization, contents filtering, and parsing. The first step accesses the raw data files and reads their contents. Tokenization step investigates the raw data catalog, which contains metadata related to the file format specification and their contents, with the purpose of verifying if the obtained raw data in the previous step correspond to the domain of the current computer simulation. As the next step, contents filtering considers the user's specification to define which contents and their values should be parsed and stored into the provenance database. Thus, this step avoids storing attributes that will not be used in domain-specific analysis. Then, parsing step converts each filtered raw data contents to present a specific data structure acceptable by the DBMS. This data structure requirement is informed by the raw data catalog.

Raw Data Indexing component indexes specific contents from files to improve direct access to specific regions of the raw data space and managing metadata associated to the dataflow. Besides the content read from raw data files and parsing, *Raw Data Indexing* has to create indexes to the accessed raw data contents according to the defined indexing algorithm. For instance, bitmap indexing [9] generates a bitmap to indicate the presence of a specific value on the raw data content, while positional indexing [7] provides the location of a value within the content of the raw data files.

The *Raw Data Extraction* component is ruled by the algebraic operator Raw presented in Section 4.3. When both components are employed (*Raw Data Extraction* and *Raw Data Indexing*), they are ruled by the algebraic operator RawI presented in Section 4.4, where the program T_i determines the chosen indexing algorithm. In a more practical perspective, considering the indexing algorithms in the literature [7–9,28], *ad-hoc* programs or third-party tools (*e.g.*, FastBit tool) can be invoked by this component to generate indexes. Thus, *Raw Data Indexing* component is highly recommended to simulations that generate a huge amount of raw data in files or whose files assume more advanced data structures, such as meshes. The definition of the most suitable strategy to be used (raw data extraction and index generation) depends on the volume of data and the data structures of the specific contents in raw data files, as discussed in Sections 3.1 and 3.2.

To instantiate new cartridges in ARMFUL, users initially have to identify to which component this cartridge corresponds to, *Raw Data Extraction* or *Raw Data Indexing* component. With the *Raw Data Extraction* component, the cartridge needs three methods to access and extract raw data elements from files, named as *extract*(), *run*(), and *access*(). The *extract*() method aims at identifying and extracting raw data elements for the raw data file attributes chosen by the user. Thus, this method assumes that all specified contents are present in the raw data file. The *run*() method is used with the *extract*() method to extract raw data from several files with the same file format. It means that the *run*() method invokes the *extract*() method for each raw data file, which returns domain-specific data to be represented in the same output data set. The *access()* method retrieves raw data extracted by *extract*() method.

To instantiate cartridges for the *Raw Data Indexing* component, the three methods to access and generate indexes for raw data in files, are named as *index*(), *run*(), and *access*(). The *index*() method generates indexes for all attributes chosen by the user, which are present in a single raw data file. Algorithm 1 implements this method, assuming that the *index*() method invokes the indexing algorithm (argument *algorithm*) chosen by the user and all specified contents (variable *cts*) are present in the raw data file. As happens with the *Raw Data Extraction* cartridge, the *run*() method invokes the *index*() method to generate indexes for all selected raw data contents in each record from multiple raw data files. The *access*() method retrieves raw data using indexes generated by *index*() method, as presented in Algorithm 2.

Algorithm 1. Index generation for accessing raw data from files

Input:
- **algorithm**: Algorithm for accessing raw data via indexes
- **path**: Path of input file
- **file$_{name}$**: Input file name
- **cts**: List of raw data contents

Output:
- **dataSet**: Output data set

```
1.  function index(algorithm, path, file_name, cts):
2.      dataSet ← DataSet(cts)
3.      if exists file_name in path then:
4.          for each element in file_name do:
5.              values_element ← { }
6.              position ← 0
7.              for each content in cts do:
8.                  ctValue ← generateIndex(algorithm, content, element)
9.                  values_element[position] ← ctValue
10.                 position++
11.             end do
12.             dataSet.addElement(values_element)
13.         end do
14.     end if
15.     return dataSet
16. end function
```

Once the cartridge is identified, users need to define programs for the methods: (i) *extract*() or *index*(), and (ii) *access*(). The *run*() method is provided by ARMFUL. To help on using the *extract*() method, ARMFUL presents a cartridge, named as *External Invoker*, which accesses and extracts the contents from raw data files by invoking *ad-hoc* programs. Therefore, using this cartridge, users need to specify the raw data file paths, the raw data contents to be extracted, and the command line to invoke the *ad-hoc* program. Since we performed experiments in CFD

simulations, Section 6 presents the invocation of an *ad-hoc* program developed with the Python API of ParaView toolkit to extract raw data from XDMF and HDF5 files.

To support raw data indexing, this first version of ARMFUL presents two cartridges for the *Raw Data Indexing* component, named as *Positional Indexing* and *FastBit tool*. The *Positional Indexing* cartridge consists of a program developed by our team to generate positional indexes by adapting the algorithm presented in the RAW approach [8]. The *FastBit tool* cartridge implements the invocation of FastBit tool (Figure 3) to generate bitmap indexes. In relation to the *access*() method implemented on those indexing cartridges, we developed from scratch a code to access the raw data via the indexes generated by the *Positional Indexing* cartridge. We use features provided by the FastBit tool to implement the *access*() method to the *FastBit tool* cartridge. As a limitation of this first version of ARMFUL, it does not consider modifications in raw data files after the extraction or indexing process (only append operations). In future versions it should implement adaptive indexing approaches. More details about ARMFUL and its components implementation can be found in its website[2].

Algorithm 2. Raw data access using indexes

Input:
- **algorithm**: Algorithm for raw data access using indexes
- **path**: Path of input file
- **file$_{name}$**: Input file name
- **dataSet**: Data set, where each element contains generated indexes
- **cts**: Raw data contents of interest

Output:
- **output**: Output data set

```
1.  function access(algorithm, path, file_name, dataSet, cts):
2.      output ← DataSet(cts)
3.      if exists file_name in path then:
4.          for each element in dataSet do:
5.              values_element ← { }
6.              position ← 0
7.              for each ctIndex in element.values do:
8.                  if dataSet.contents[position] in cts:
9.                      ctValue ← getValueByIndex(method, ctIndex)
10.                     values_element ← values + {ctValue}
11.                 end if
12.                 position++
13.             end do
14.             output.addElement(values_element)
15.         end do
16.     end if
17.     return output
18. end function
```

5.2. A PROV-Compliant Provenance Database Component

Extracted raw data has to be loaded into a database to be queried and analyzed. In ARFMUL, this database is a provenance database that integrates provenance data, extracted raw data and execution data. The *Provenance Database* component is this provenance database that can be analyzed through the *Query Processing* component.

The *Provenance Database* representation follows the dataflow definition presented in Section 4.1, which allows for querying data transformations and data sets involved in computer simulations uniformly in the provenance database. It extends the PROV-Df [16] data model (see Figure 5), which is a W3C PROV-DM[3] specialization data model, to represent metadata about the data transformation and the flow of data sets. More specifically, the extended PROV-Df represents properties about the scientific workflows and their associated dataflow generation at the logical level, *i.e.*, data element flow considering indexing. This data model describes a scientific workflow in relation to the simulation programs (*i.e.*, workflow activities), their invocations, and data dependencies (*i.e.*, attributes consumed, produced, and propagated represented in data sets), while it also gathers

[2] https://hpcdb.github.io/armful
[3] https://www.w3.org/TR/prov-dm

provenance data related to the execution of the scientific workflow, such as metadata about the execution of a specific activation (or task) and their raw data elements extracted or indexed from files.

The original version of PROV-Df is composed of three types of classes: dataflow concepts (white classes in the UML class diagram with exception of entities *Workflow*, *Activity*, and *Relation*), dataflow generation during the execution of a scientific workflow (dark gray classes) and environment configuration (light gray classes). Each class in PROV-Df is extended from the core and extended structures of PROV standard (entity, agent, plan and activities). The stereotypes in the UML class diagram are used to represent PROV components regarding the PROV-DM specification. For more detail about PROV-Df, please refer to [16].

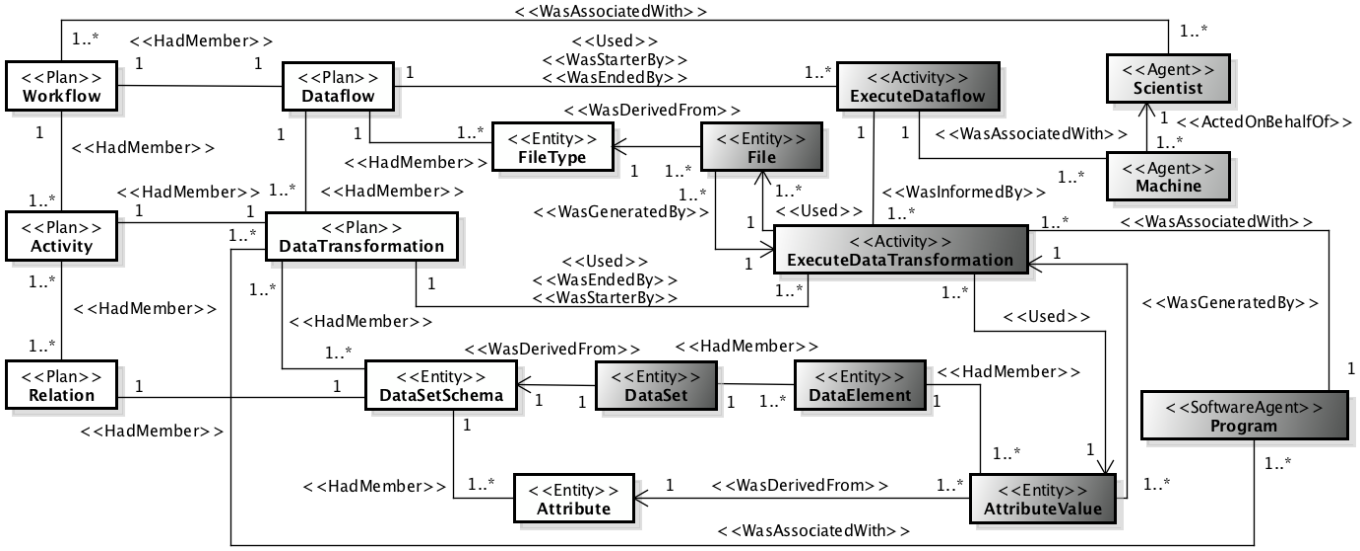


Figure 5. PROV-Df data model with extensions for scientific workflow concepts [16].

Differently from the original version of PROV-Df, we changed the entities *Workflow*, *Activity*, and *Relation*, as presented in Figure 5. These changes added conceptual descriptions about computer simulation modeling, since dataflow entities are only dedicated to the data generation during the execution of computer simulations, while workflow concepts introduce metadata about the specification of simulation programs (*e.g.*, program invocation) and their rules (*e.g.*, data dependency between data transformations). Considering the improvements for raw data extraction and index generation, this process handles data transformations (entity *ExecuteDataTransformation*) ruled by the Raw or RawI operator addressing consumed raw data files (entity *File*) and raw data extracted or indexed (entity *AttributeValue*) to a specific attribute (entity *Attribute*). Meanwhile, a set of attributes is specified in a data set schema (entity *DataSetSchema*).

We validated PROV-Df compliance with W3C PROV-DM using ProvToolbox[4] to generate files according to the PROV-DM specification (in PROV-N, JSON, XML, and DOT file formats) and ProvValidator[5] web service [37] to analyze these generated files. Our focus is on the runtime dataflow analysis. Therefore PROV-Df represents provenance and domain specific data ready to be queried as they are being generated by the SWMS. In ARMFUL, PROV-Df is mapped to a relational database to represent the dataflows and their data transformations, as shown in [38]. A relational DBMS is used to query raw data in a structured way, thus easing user's analysis. Users are able to submit queries to the DBMS to get information about the entire dataflow and its execution (such as elapsed time and errors from the execution of simulation programs). In addition, differently of existing solutions for raw data file analysis, users have access through provenance database to selected raw data files and their contents among a large number of generated files spread in several workspaces. The reference to files is registered in the provenance database as pointers (*e.g.*, Uniform Resource Identifier, known as URI).

For all workflow activities ruled by Raw or RawI operator (entity *Activity*), the output relation (entity *Relation*) presents the domain-specific data values extracted or the generated indexes, respectively, from the raw data files or propagated from previous workflow activities. Thus, users can elaborate deeper analytical queries on the provenance database, involving workflow composition (the chaining of simulation programs), dataflow information and raw data from/within files.

[4] http://lucmoreau.github.io/ProvToolbox

[5] https://provenance.ecs.soton.ac.uk/validator/view/validator.html

5.3. Query Processing Component

The *Query Processing* component aims at providing a mechanism to query provenance and raw data stored in a provenance database following our data model (Section 5.2). This component varies its behavior according to the chosen strategy for extracting or indexing raw data from files. If it is only considered the raw data extraction, the obtained data elements from files are stored in a database respecting the provenance data model and the specifications of the DBMS. Therefore, this component only allows for queries that are supported by the DBMS query processor, since all raw data contents are stored in its provenance database.

Depending on the strategy for raw data extraction and index generation, query processing presents additional steps related to raw data file access and data retrieval, as discussed in Section 3.2. If it is assumed a strategy that applies indexing techniques, provenance database presents the generated indexes (or index files), instead of the raw data elements (with their attribute values). In this case, this component has to retrieve the generated indexes (or index files) from the provenance database, to use them to access (and retrieve) the raw data contents in file. As presented in the example of Figure 3, FastBit tool can be used to access raw data from files using the generated indexes (*i.e.*, bitmap files).

The *Query Processing* component of ARMFUL uses the *access*() method of cartridges from the *Raw Data Extraction* and *Raw Data Indexing* components to access raw data from files during the workflow execution. To explain the use of *access*() method for dataflow analysis, let us consider a theoretical example. We assume two workflow activities, *A* and *B*, where *B* depends on *A* (*i.e.*, output data set O_S from activity *A* is consumed by activity *B*). Activity *A* is ruled by the RawI operator using a data access method via indexes. In this case, the invocation of activity *B* needs the values for each element from data set O_S produced by activity *A*. Then, it is necessary that the *Query Processing* component considers the generated indexes for each data element (from data set O_S) in activity *A* and use them for retrieving raw data from files with the *access*() method.

5.4. Coupling ARMFUL to a SWMS

To couple ARMFUL to a SWMS, the workflow definition has to be instrumented to call its components for provenance gathering and raw data extraction/indexing. To support raw data extraction and indexing, each workflow activity that has a data transformation has to include one raw data transformation. Raw data transformations, ruled by Raw or RawI operators, are decoupled from the workflow activities, but keep the consistency with all other data-centric transformations.

Raw data file parsing requires the invocation of *ad-hoc* programs or third-party tools (similar to the specification of simulation programs). This way, to add raw data extraction or indexing to a specific data transformation activity, the workflow definition has to include another data transformation activity, which is ruled by Raw (for raw data extraction) or RawI (for raw data indexing) operators (see Sections 4.3 and 4.4). It is also important to highlight that the SWMS has to manage the path reference of the raw data files and users have to define which attributes should be tracked by the raw data extraction or indexing process. Although Raw and RawI operators are fairly compatible with Chiron's algebra [13], they work in an isolated way in other SWMS, provided that the data being gathered can be stored in relational tables with data provenance.

To bind ARMFUL components with a SWMS, we provide three Java ARchive (JAR) files, named as *ProvenanceGatherer*, *DataIngestor*, and *RawDataAnalyzer*. *ProvenanceGatherer* JAR file aims at managing provenance data gathered during the execution of scientific workflows. This program should be used when the SWMS does not support provenance data management. However, if the SWMS manages provenance data, it needs to follow the PROV-Df data model to be coupled to ARMFUL. In our approach, A-Chiron already presents a provenance database that follows PROV-Df data model. The *DataIngestor* JAR file initializes a database server using a column-based DBMS, MonetDB [39], to store the provenance data gathered by the *ProvenanceGatherer* program. As the *ProvenanceGatherer* program, the *DataIngestor* program is only recommended if the SWMS does not support provenance data management.

The *RawDataAnalyzer* JAR file presents two different methods for accessing raw data from files: raw data extraction and indexing. The former method extracts the contents from raw data files, while the last one accesses and generates indexes to the data elements in raw data files. All cartridges for raw data extraction and indexing presented in Section 5.1 were developed with the *RawDataAnalyzer* program. Therefore, to couple a SWMS to ARMFUL, we only need to invoke the *RawDataAnalyzer* program, defining the raw data access cartridge.

6. Experimental Evaluation

In this paper, we couple ARMFUL to Chiron SWMS, which we named as A-Chiron. As the first step on this development, we implemented the Raw and RawI operators in Chiron by the invocation of the raw data extraction and indexing cartridges from ARMFUL. In relation to the provenance data management, we extended Chiron to

manage extracted data and the generated indexes and, consequently, to present a provenance database consistent with our PROV-Df data model. In this section, we evaluate A-Chiron for extracting and indexing raw data during the execution of data-intensive scientific workflows. Our experiments are based on a scientific workflow in Computational Fluid Dynamics (CFD) domain [17] (see Section 6.1). Although our experimental evaluation considers a specific CFD workflow, workflows from other domains, like those in astronomy [16], geophysics [17] and bioinformatics [40], can also benefit from the raw data analysis support of ARMFUL. In the following experimental evaluation, we compare the cost with and without raw data file analysis, considering two strategies that apply indexing techniques, and present the analytical capabilities enhanced using our proposed approach.

6.1. Case Study: Computational Fluid Dynamics Workflow

In our experiments, we used as case study a workflow for analyzing the flow of an incompressible fluid on a cavity problem introduced in Section 2. This case study uses EdgeCFD software and, consequently, the modeled workflow presents the same name (*i.e.*, EdgeCFD workflow). EdgeCFD is a Fortran90 finite element application where the kernel of the computational solution consists of a fully implicit predictor – multicorrector time integration scheme as described in [18]. The generalized trapezoidal rule is employed in the time discretization, with an adaptive time stepping procedure based on a proportional-integral-derivative (PID) controller.

The nonlinearities due to the convective term on the Navier–Stokes equation (describes the motion of viscous flow) are treated by an Inexact Newton–GMRES algorithm [17]. In this solution algorithm, at the beginning of the nonlinear iterations in each time step, the algorithm computes large linear tolerances, producing fast nonlinear steps. As the iterations progress towards the solution, the inexact nonlinear method adapts the GMRES tolerances to reach the desired accuracy. The tolerance, which the linearized system is solved, known as the forcing term, plays an instrumental role in the numerical performance of this method. To enhance the robustness of these methods, they could be employed with some globalization strategies, which contemplate auxiliary procedures that increase the convergence to the solution when good initial approximate solutions are not available. Four forcing terms are implemented in EdgeCFD and its applicability will be analyzed in this study. Moreover, we choose this workflow as a case study, since their scenarios manipulate huge volumes of data. This data may be stored as files in different formats and assume non-trivial data structures for generating indexes. To facilitate these issues, we used some existing solutions, such as ParaView toolkit and FastBit tool, for accessing raw data from these heterogeneous files and generating indexes.

Figure 6 presents the EdgeCFD workflow using the workflow algebra described in Section 4, where <<*stereotypes*>> represent algebraic operators that describe the activity behavior. In this workflow, only activities *EdgeCFD Pre* and *EdgeCFD Solver* use binaries from EdgeCFD software (*i.e.*, *edgecfdPre* and *edgecfdSolver* programs, respectively), while other activities without raw data extraction and index generation support run programs (*uncompressDataset*, *preProcessing*, and *setSolverConfiguration*) to lead with compressed datasets, configuration files for executing EdgeCFD binaries, and raw data stored on specific file formats (*e.g.*, HDF5 files). *EdgeCFD Pre* consumes an input data set (*i.e.*, mesh) and defines the properties of the mesh (*i.e.*, partitioning for parallel processing and nodal reordering for improving data locality) to be employed on the CFD solver (activity *EdgeCFD Solver*). This way, *EdgeCFD Solver* consumes the generated files by *EdgeCFD Pre* and computes the behavior of a fluid in a specific mesh along a predefined time interval. As a result, this solver produces different files in XDMF and HDF5 file formats, which present information about the pressure, velocity and other fluid properties in that time interval. More specifically, HDF5 files contain the values for domain-specific attributes (*e.g.*, fluid velocity) and XDMF files present some pointers to address each HDF5 file for a specific time steps. *Pre RDFA* and *Solver RDFA* are activities with raw data file analysis support, which we use to vary the strategy for raw data extraction and index generation. For this reason, we have two possible algebraic operators for those activities. As presented in previous sections, activities with raw data file analysis (*i.e.*, *Pre RDFA* and *Solver RDFA*) may extract or index the contents of those files by Raw and RawI operators, respectively.

Users normally perform this computer simulation with the purpose of analyzing the behavior of the fluid over time, which requires raw data analysis from XDMF and HDF5 files. However, due to the huge volume of data from those files, the time overhead for storing raw data into a provenance database may be higher than the execution of simulation programs [8] itself, when it is considered a total raw data extraction. In these circumstances, users extract only a specific region of interest to analyze it (*i.e.*, partial extraction), such as a line of points for the cavity problem, considering the boundary conditions and Reynolds number (a non-dimensional number characterizing fluid flow) presented in Figure 7(a). This specific region collects velocity components to be compared with existing reference solutions (Lo *et al.* [41]), that is, controlling solution accuracy for a given set of parameters. As results, Figure 7(b) presents the horizontal and vertical velocities represented by attributes V_x and V_z, respectively. In our experiments, we used the ParaView toolkit with Python API for accessing raw data from the region of interest (*i.e.*, region of the generated meshes) in XDMF and HDF5 file formats after the execution of activity *EdgeCFD Solver*.

As an input to this Python API, users have to provide the raw data file path and the attributes to be extracted from the files. Furthermore, we store this selected data into a tabular file format, which is similar to the CSV file format.

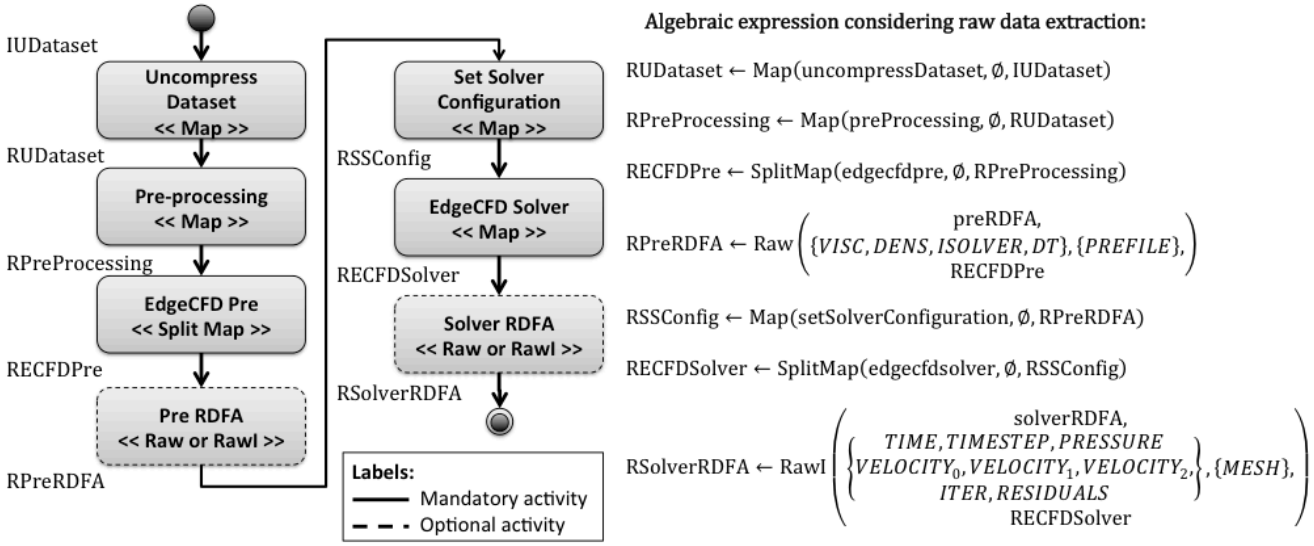


Figure 6. The EdgeCFD workflow specification with associated algebraic expressions for raw data extraction.

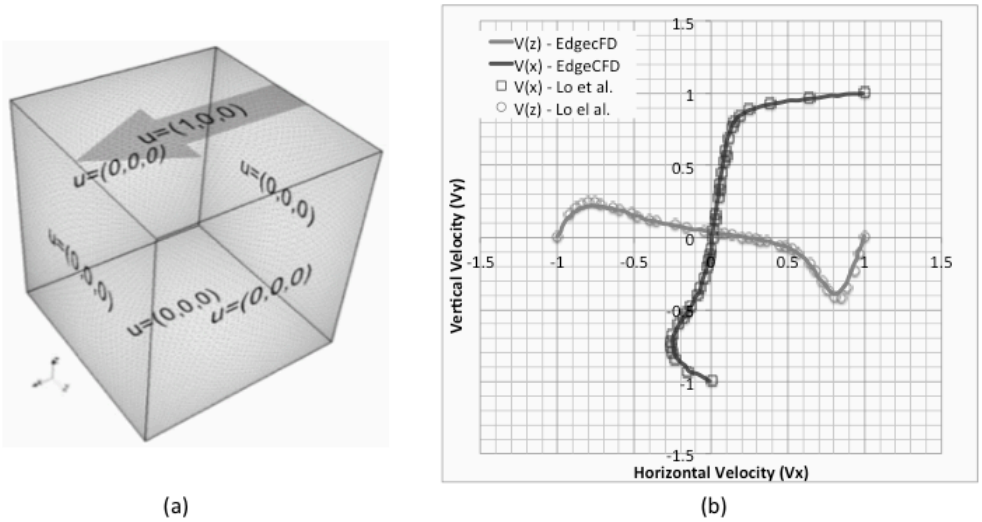


Figure 7. EdgeCFD simulation: (a) boundary conditions and (b) cavity flow validation for Reynolds number Re=1000.

Besides varying CFD workflow specifications, users often need to try some alternative values for their input attributes, named as parameter sweep [42]. In CFD simulations, users commonly vary the mesh size, fluid density, fluid viscosity, and Inexact Newton solver parameters, such as forcing terms and backtracking globalization strategy parameters. When the physical domain is discretized by linear simplexes (tetrahedra), as the mesh size grows, the number of vertices and tetrahedra also grows. Note that mesh sizes in a real scenario may be as big as several billions of vertices and tetrahedra. A tetrahedron contains four integer numbers (connectivity) to map the vertices and another integer for the tetrahedron identifier. A finite element mesh (represented in HDF5 files) contains the coordinates in three dimensions of all points, and the connectivity. The outputs are the fluid pressure, and the fluid velocity in three components (x, y, and z) at given time steps, also stored in HDF5 format. In our experiments, we used three types of input mesh: coarse, medium and fine meshes. The coarse mesh presents 32,109 vertices and 128,436 tetrahedra; the medium mesh presents 257,852 vertices and 1,031,408 tetrahedra, and fine mesh presents 1,145,569 vertices and 4,583,276 tetrahedra.

6.2. Alternatives of the EdgeCFD Workflow using A-Chiron

In this paper, we developed different cartridges for A-Chiron, considering raw data file analysis from a specific region of interest (*e.g.*, a line of coordinates with domain-specific data) in a CFD simulation, as described in Section 6.1. We only modified activity *EdgeCFD Solver* to support different strategies for raw data extraction and indexing, since this activity manipulates the most part of the data generated (HDF5 files) by the EdgeCFD

workflow execution and presents the most advanced data structures (*i.e.*, meshes). Thus, the following set of cartridges were developed and used in our experiments:

- ***ParaView*:** development of an *ad-hoc* program that accesses raw data in files and extracts them. This program uses ParaView toolkit with Python API to extract raw data from XDMF and HDF5 files;
- ***FastBit tool*:** use of FastBit tool to generate bitmap indexes of raw data in XDMF and HDF5 files. To access raw data on these file formats, we use ParaView toolkit with Python API;
- ***Positional Indexing*:** development of a program to generate positional indexes from raw data in XDMF and HDF5 files, which was based on the positional indexing algorithm of RAW [8]. To access raw data on these file formats, we also use ParaView toolkit with Python API.

Considering the aforementioned cartridges, we developed different versions of EdgeCFD workflow to evaluate the support for raw data extraction and indexing, when we vary the algorithm. Figure 6 presents two workflow activities, named as *Pre RDFA* and *Solver RDFA*, that employ raw data file analysis. Those activities were defined as optional activities, since they were disabled (or removed) according to the raw data file analysis support. Thus, the following versions were proposed for the EdgeCFD workflow:

- **Baseline:** Workflow version without raw data file analysis support. Thus, activities *Pre RDFA* and *Solver RDFA* are not present in workflow specification. In addition, this workflow version do not support query processing with raw data elements;
- **Raw Data Extraction (named as rde):** Workflow version with both activities for raw data analysis using the cartridge *ParaView*. Thus, this workflow version enables the extraction of raw data from XDMF and HDF5 files (*i.e.*, without raw data indexing);
- **Bitmap Indexing (named as idx-bitmap):** Workflow version with both activities for raw data analysis. More specifically, this version uses Paraview toolkit and FastBit tool (cartridges *ParaView* and *FastBit tool*, respectively), which enables accessing raw data from XDMF and HDF5 files and generating bitmap indexes from the accessed contents, respectively; and
- **Positional Indexing (named as idx-positional):** Workflow version with both activities for raw data analysis. Furthermore, this workflow version uses ParaView toolkit and a program for positional indexing, which enables the access of raw data from XDMF and HDF5 files and the generation of positional indexes following the RAW's implementation presented in [8], respectively. It means that this workflow version uses cartridges *ParaView* and *Positional Indexing*.

6.3. Environment setup

All experiments presented in this paper were executed in Stampede cluster, a Dell Linux Cluster based on 6400+ Dell PowerEdge server nodes, each outfitted with 2 Intel Xeon E5 (Sandy Bridge) processors and an Intel Xeon Phi Coprocessor (MIC Architecture) at the Texas Advanced Computing Center (TACC).[6] Each compute node contains 32GB of RAM memory with an additional 8GB of memory on the Xeon Phi coprocessor card. Nodes are interconnected with Mellanox FDR InfiniBand technology in a 2-level (cores and leafs) fat-tree topology. All experiments performed in this paper used 240 cores considering, exclusively, the Intel Xeon E5 processors. In this cluster architecture, we instantiate a provenance database using the PostgreSQL DBMS in one of the available computational nodes to store provenance and raw data into the DBMS. This database node uses just 16 cores. Chiron connects its master node to the DBMS node to get activities ready for execution and manage provenance data. In Chiron's code, this connection between the workflow engine and the DBMS is performed by a JDBC driver to the PostgreSQL.

6.4. Experimental results

We performed 5 experimental evaluations using EdgeCFD workflow (see Section 6.1) with the following goals:

- **Time cost according to the amount of raw data extracted from files (named as EXTRACTION):** Measurement of the time cost for extracting the contents from XDMF and HDF5 files considering two granularity levels. For the granularity levels for raw data extraction, we considered partial – considering a region of interest as discussed in Section 6.1 – and total extraction of raw data from files.

[6] https://www.tacc.utexas.edu

Since we only consider raw data extraction in this experimental evaluation, we used the workflow version *rde*. However, we also compare this workflow version using different granularity levels (partial and total) with the workflow version *baseline* (*i.e.*, without raw data extraction).

- **Time cost for raw data file analysis (named as RDFA)**: Measurement of the time cost for extracting or indexing raw data (*i.e.*, process named as Raw Data File Analysis or RDFA) with the four versions of EdgeCFD workflow using the fine mesh (presented in Section 6.1) as input data. We also compare this elapsed time with other time costs, which are associated to store data into provenance database (provenance and raw data loading into database) and execute workflow activities;
- **Workload analysis for indexing raw data from files (named as INDEXING):** Measurement of the time cost to generate indexes of raw data accessed from files with different workloads. Each workload considers a specific mesh size in input data set. Each workflow version was executed three times, varying the input data set, which assumed the following mesh sizes for analyzing fluid dynamics: coarse, medium, and fine meshes (see Section 6.1);
- **Workload analysis for data ingestion into provenance database (named as INGESTION)**: Evaluation of the time cost to store provenance and raw data into provenance database with different workloads, similar to the experimental evaluation INDEXING;
- **Query processing (named as QUERY)**: Analysis of the raw data query capabilities. We present queries to analyze domain-specific content (from files); specific related elements from multiple raw data files; and performance execution data.

EXTRACTION. We modeled and executed EdgeCFD workflow with and without raw data extraction as presented in Section 6.2 (*baseline* and *rde*, respectively). In this experiment, we considered 4,800 alternatives of attribute values (*i.e.*, forcing terms and backtracking globalization strategy, and fluid viscosity), which manipulated 86,400 HDF5 and XDMF files and required approximately 1.09 TB of storage space. As input data set, it was considered the fine mesh. To measure the time cost associated to store the contents from raw data files into a DBMS, we compare the average elapsed time from 3 executions of each workflow version.

In this case, we considered the following workflow versions: *baseline*, *rde* with total extraction, and *rde* with partial extraction. Table 1 presents the elapsed time for the execution of workflow activities, raw data extraction, and data ingestion into DBMS and the total workflow elapsed time. We do not consider the contribution of index generation, since no indexing algorithm was applied in this experiment. As expected, the elapsed time for executing workflow activities does not present a high variance in those results, since we do not change the complexity of simulation model or the mesh size for the CFD simulation. The scheduling algorithms, implemented in Chiron for parallel workflow execution, can also explain the variance between elapsed times for workflow activities.

Table 1. Workflow elapsed time associated to the granularity level for raw data extraction.

Workflow version	Total elapsed time in minutes				
	Execution of workflow activities	Raw data extraction	Data ingestion into DBMS	Workflow	Standard deviation for workflow time
baseline	37.22	0.00	2.58	39.80	1.35
***rde* with partial extraction**	36.72	2.36	16.52	55.60	3.04
***rde* with total extraction**	41.03	6.21	84.36	131.60	3.66

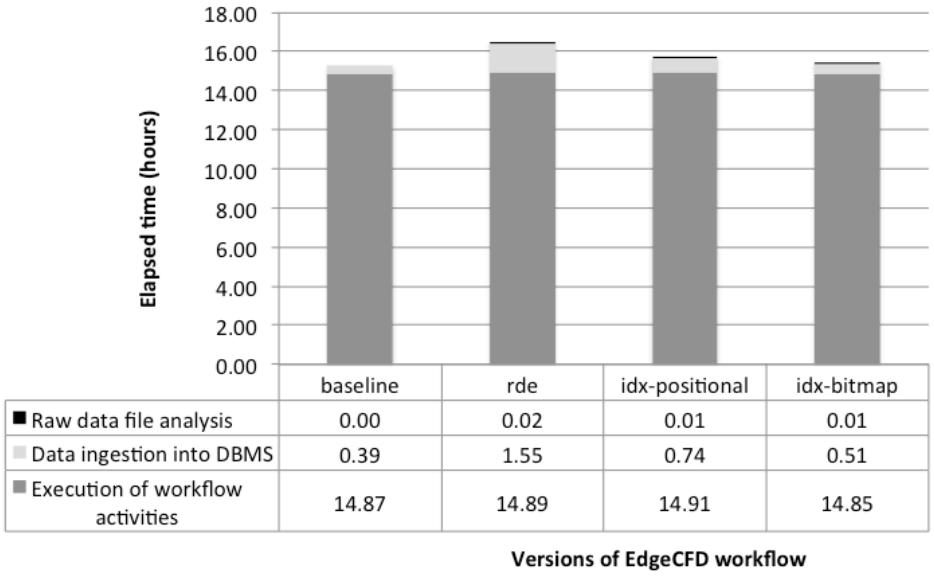
Still considering the obtained results for this experiment, when we increase the volume of data extracted from files (from partial to total extraction), the elapsed time for raw data extraction also increases. In consequence, the elapsed time for data storing in the database also increases, since more data have to be loaded into Chiron's provenance database. In CFD domain, users normally analyze some predefined positions in the mesh, such as the centerline for a specific Reynolds number attribute value as discussed in Section 6.1, to perform their CFD investigations. Considering the execution of a program for extracting this region of interest, it introduces approximately 2.36 minutes to the workflow elapsed time. For extracting raw data from XDMF and HDF5 files, our program is a Python script that uses ParaView's Python API. We can also state that raw data extraction elapsed time is negligible for all workflow versions, *i.e.*, 4.72% of workflow elapsed time in the worst case.

Comparing the workflow elapsed time, we note that workflow *rde* with partial extraction presents a workflow elapsed time overhead equals to 39.70% in relation to the *baseline* (*i.e.*, without RDFA), while this overhead is equal to 330.65% for the *rde* with total extraction. For this reason, other experiments from this paper only consider

EdgeCFD workflow with partial extraction, when we perform RDFA. Despite of the aforementioned overhead for data storing, partial extraction still presents several benefits for domain-specific analyses. As discussed before, this analytical capability at runtime provides to the users some features to decide the future of their computer simulations. For instance, they may abort a workflow execution or fine-tune some attribute values according to their algorithm convergence (as discussed in example from Figure 1).

RDFA. In this experiment, we executed EdgeCFD workflow to analyze the elapsed time for: the execution of workflow activities; raw data file analysis (encapsulate elapsed times for extracting raw data and generating indexes); and storing data into DBMS. During the execution, we vary the approach for raw data file analysis, considering the 4 workflow versions presented in Section 6.2. More specifically, this workflow was configured to use the fine mesh with the purpose of representing a real scenario in CFD domain.

Figure 8 presents the workflow elapsed time of the execution of the workflow activities, raw data file analysis, and data ingestion for each workflow version. With those results, we note that workflow version *rde* needs 72.20 minutes more than *baseline* (*i.e.*, without RDFA), and 24.20 minutes in relation to the worst workflow elapsed time using an indexing algorithm (*i.e.*, *idx-positional*). With *rde*, we can observe a data storing overhead equals to 69.75 minutes in relation to the *baseline*, while the positional and bitmap indexing approaches present overheads equal to 20.91 and 7.37 minutes, respectively. In addition, we note that the cost for running indexing algorithms in the worst case is approximately 79 seconds, which represents 0.13% of the total workflow elapsed time. Therefore, we conclude that storing data into DBMS is a severe bottleneck, even with partial raw data extraction. Meanwhile, we overview that index generation may reduce the data storing and do not interfere significantly in workflow elapsed time. We also emphasize that this better performance for raw data indexing is associated to the data structures from attributes in the analyzed raw data files, *i.e.*, meshes in XDMF and HDF5 files.



	baseline	rde	idx-positional	idx-bitmap
Raw data file analysis	0.00	0.02	0.01	0.01
Data ingestion into DBMS	0.39	1.55	0.74	0.51
Execution of workflow activities	14.87	14.89	14.91	14.85

Figure 8. Raw data file analysis cost using fine meshes.

INDEXING. This experiment evaluates the index generation, when we vary the number of vertices in the input mesh (*i.e.*, mesh size) for the EdgeCFD workflow. So, we analyze the sequential elapsed time for generating indexes when we vary the volume of raw data analyzed in files (mesh size) and the indexing algorithm (positional indexes and bitmap indexes). Although workflow version *rde* do not generate indexes using an external program, raw data extracted from files is stored in DBMS. In this circumstance, the gathered provenance data have to be represented with raw data in the same data model, PROV-Df. So, DBMS has to parse the extracted raw data and generate internal indexes to correlate provenance data, raw data files, and extracted raw data. If we employ indexing algorithms, such as bitmap or positional for raw data contents from files, the time cost for generating internal indexes in DBMS is aggregated with to time cost of generating indexes to the access raw data contents in files. In addition, since the *baseline* do not present raw data extracted from files, they do not generate indexes via a DBMS and, consequently, the sequential elapsed time is equal to zero for raw data indexing.

Considering the experimental results presented in Figure 9, we note that *rde* has the highest sequential elapsed time for indexing raw data, when it is compared to the indexing algorithms. For instance, this approach is approximately 2 times slower than the positional indexing algorithm, when we consider fine meshes similar to real scenario for CFD simulations. More specifically, we note reduction of 64.95% and 46.39% in time overheads for

generating indexes, when we used a positional and bitmap indexing algorithms, respectively, instead of an algorithm that extracts raw data.

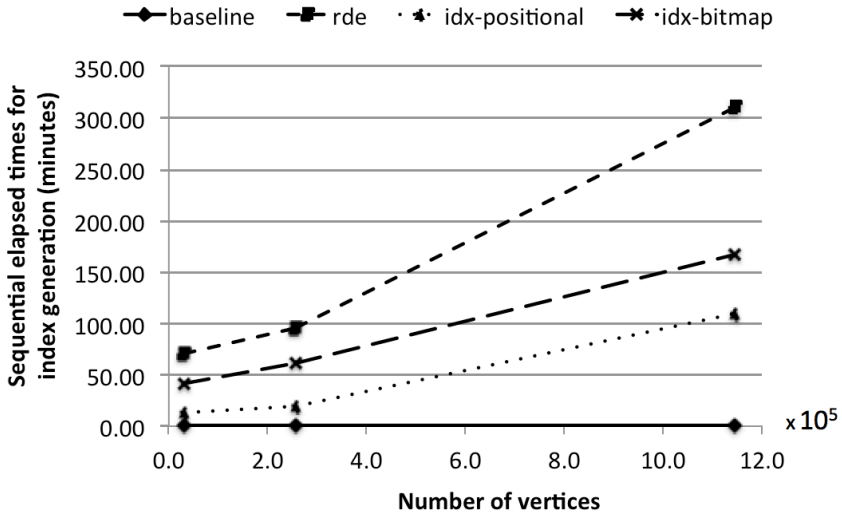


Figure 9. Sequential elapsed time for generating indexes with different workloads.

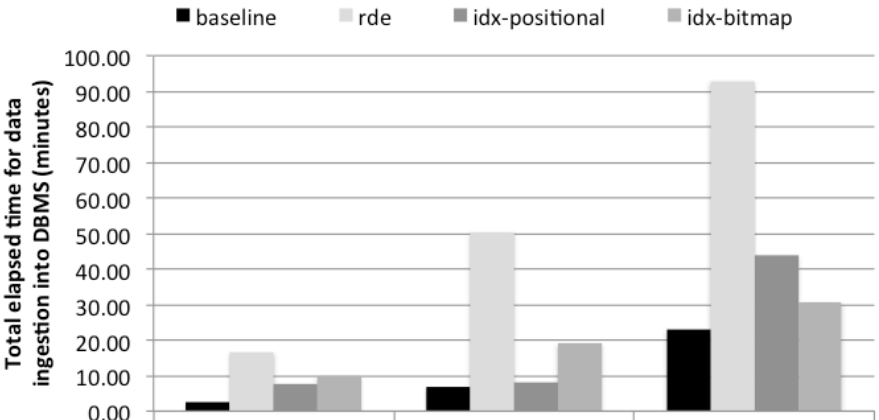
If we isolate our analysis only for bitmap and positional indexing algorithms (*idx-bitmap* and *idx-positional*, respectively), we noted that bitmap indexing algorithms, which we use FastBit tool, presents the performance worse than positional indexing algorithm. Due to the many distinct values assumed by the indexed attributes in output raw data files from EdgeCFD solver, bitmap indexing algorithm generates sparse bitmap arrays (or bitmaps) that means arrays with a large number of columns and many zero values (non-occurrence of a specific attribute value). Consequently, this indexing algorithm takes more time to generate indexes and requires more storage space. We observe these issues in Figure 9 and when the positional indexing algorithm consumes 4.4 GB less than the bitmap indexing algorithm with fine meshes in file system, disregarding storage space from provenance database. Although FastBit tool enables users to modify the indexing algorithm, we did not consider these advanced configurations of FastBit in our experiments. Therefore, we could reduce elapsed time for index generation using FastBit tool, if we generate bitmaps based on some intervals of values (based on inequalities, discussed in Section 3.2), instead of specific attribute values (based on equations, *e.g.*, attribute *pressure = 0.002*).

INGESTION. This experiment evaluates the data ingestion into DBMS, when we vary the number of vertices in the input mesh (*i.e.*, mesh size) for the EdgeCFD workflow. The workloads of this experiment are similar to the experiment INDEXING. Moreover, for each mesh size, we analyze the elapsed time for ingesting data into the provenance database, varying the raw data extraction support and the indexing algorithm (positional indexes and bitmap indexes).

Figure 10 presents the obtained results considering the 4 versions for EdgeCFD workflow. Firstly, we note that data ingestion cost increases as well as we increase the mesh size, since more provenance, execution, and raw data are stored in Chiron's provenance database. In all mesh sizes, workflow version *rde* presents the highest data ingestion overhead, since it accesses the contents from raw data files, parses them to a specific data structure, and loads this structured data into the provenance database. Different from this workflow version, the approaches with indexing algorithms present optimized data structures to address raw data (*e.g.*, integers for positional indexes), as discussed in Section 3.2 and experiment INDEXING. This experimental result also demonstrates a reduction of 52.55% in time overhead for data ingestion into the Chiron's DBMS, when we used a positional indexing algorithm (44.11 minutes) in this workflow with fine meshes, instead of an algorithm limited to the raw data extraction (92.95 minutes).

With the *idx-bitmap* implemented in FastBit tool, we decide to load only the path for index file (*i.e.*, catalog that can be used for query processing) generated by this tool to reduce the elapsed time for data ingestion, as observed in our experimental result. Moreover, this developed cartridge with FastBit tool requires less storage space for provenance database than the algorithm for positional indexing (please refer to Table 2). With fine meshes, our database requires 3.20 GB of storage space in the workflow version *baseline*, 6.40 GB in *rde*, 3.20 GB in *idx-bitmap*, and 5.73 GB in *idx-positional*. Although bitmap indexing algorithm presents a reduced data ingestion overhead, it takes more time to generate indexes in relation to the positional indexing algorithm (significant overhead in relation to the positional indexing algorithm, as presented in Figure 9) and to query raw data, since the stored index files need to be accessed into provenance database and use by FastBit's query processor to access the raw data from files. Differently, positional indexing algorithm only uses the stored positional indexes

into provenance database for accessing raw data from files. Those issues for querying raw data from files are discussed in more detail in experiment QUERY.



	Coarse	Medium	Fine
baseline	2.58	6.85	23.20
rde	16.52	50.37	92.95
idx-positional	7.75	8.21	44.11
idx-bitmap	9.96	19.16	30.57

Figure 10. Data ingestion cost varying mesh size.

Table 2. Storage space of our provenance database for the EdgeCFD workflow varying the mesh size in input data set.

Workflow version	Storage space in GB		
	Coarse meshes	Medium meshes	Fine meshes
baseline	2.88	2.98	3.20
rde	4.80	5.76	6.40
idx-positional	5.44	5.56	5.73
idx-bitmap	3.01	3.05	3.20

QUERY. EdgeCFD workflow retrieves domain-specific data from raw data files. We now focus on the workflow versions with RDFA (*rde*, *idx-bitmap*, *idx-positional*) to analyze the potential of dataflow queries to debug, fine tune the dataflow execution and analyze some domain-specific data. All queries were performed on relations (in provenance database) that contain raw data analyzed from files. We executed 3 SQL queries to consider the elapsed time of a workflow activity and the values of some attributes to analyze domain-specific data from multiple related raw data files (*i.e.*, management of the data element flow).

In the first query, named as ELEMENT_FLOW (see Figure 11), we identify multiple values of structured data in different meshes after EdgeCFD solver execution with an average velocity in *x* coordinate (attribute *VELOCITY_0* in relation *RSolverRDFA* from Figure 6) greater than 0.40 and less than 0.50 for different values of Inexact Newton solver parameter (attribute *ISOLVER*). We also relate those meshes to the fluid properties (attributes viscosity and density). While the density (attribute *DENS*) and viscosity (attribute *VISC*) are set to a certain value (*e.g.*, 1.00 and 0.001, respectively), different values are shown for the Inexact Newton parameters. Thus, this analysis investigates the contents of meshes in CFD simulation considering the data element flow between different data transformations (*Pre RDFA*, *Set Solver Configuration*, *EdgeCFD Solver* and *Solver RDFA*).

Based on this definition, users are able to restrict their analyses for relevant meshes and to understand the fluid velocity in *x* coordinate over the time, considering specific values of viscosity and density, while they vary forcing terms and backtracking globalization strategy (attribute *ISOLVER*). Figure 12 presents the table from the relational provenance database that contains domain-specific data analyzed by this query. More specifically, tables *RPreRDFA* and *RSolverRDFA* contains raw data contents extracted from files using the algebraic operator Raw or RawI (presented in Section 4.3). Moreover, this figure indicates by dashed arrows the relationships between tables, which are ensured by the invocation of identifiers of simulation programs (or tasks), *i.e.*, attributes *previoustaskid* and *nexttaskid*. Regarding those relationships, this query specified by the user is able to investigate the data element flow.

```
SELECT rprerdfa.ISOLVER, rsolvrdfa.timestep, AVERAGE(rsolvrdfa.velocity_0) as vx
FROM RPreRDFA rprerdfa, RSSConfig rconf, RECFDSolver rsolv, RSolverRDFA rsolvrdfa
WHERE rprerdfa.nextActivationID = rconf.previousActivationID
AND rconf.nextActivationID = rsolv.previousActivationID
AND rsolv.nextActivationID = rsolvrdfa.previousActivationID
AND vx>0.40 AND vx<0.50 AND rprerdfa.DENS=1.00 AND rprerdfa.VISC=0.001
GROUP BY rprerdfa.ISOLVER, rsolvrdfa.timestep;
```

Figure 11. ELEMENT_FLOW - Analysis of the fluid velocity in *x* coordinate varying the method of Inexact Newton solver.

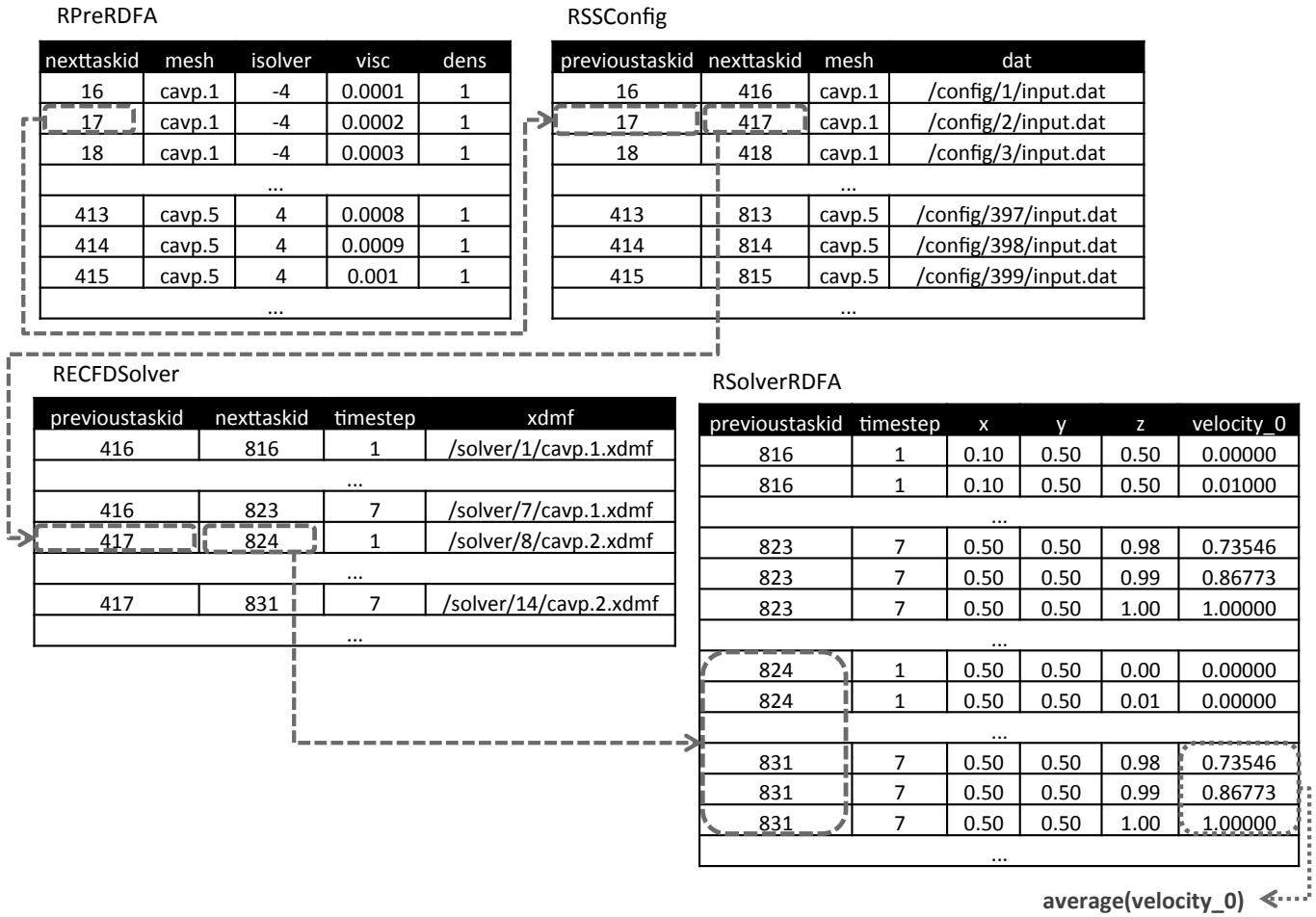


Figure 12. Tables from relational provenance database and the data element flow investigated in the query ELEMENT_FLOW.

As a second query, named as DATAFLOW (see Figure 13), we select three scalars that represent the convergence of the CFD solver execution (attributes in relation *RSolverRDFA*) considering distinct values for the fluid viscosity, while keeping fixed the fluid density, a particular setting of the Inexact Newton solver, and the time of CFD solver (*i.e.*, time step). Similarly to the query ELEMENT_FLOW, this kind of analysis allows for the users to relate workflow's output data elements, while it also represent the convergence of the CFD model (attributes *ITER* and *RESIDUALS*, similar to the Figure 1) with the input data elements, which contains fluid properties like fluid viscosity (attribute *VISC*) and density (attribute *DENS*) or solver parameters (attribute *ISOLVER*). Moreover, this query results provide a snapshot about the CFD solver execution, since it fixes a specific moment in time (attribute *TIME*), as presented in Figure 14.

```
SELECT rprerdfa.VISC, rsolvrdfa.ITER, rsolvrdfa.RESIDUALS
FROM RPreRDFA rprerdfa, RSSConfig rconf, RECFDSolver rsolv, RSolverRDFA rsolvrdfa
WHERE rprerdfa.nextActivationID = rconf.previousActivationID
AND rconf.nextActivationID = rsolv.previousActivationID
AND rsolv.nextActivationID = rsolvrdfa.previousActivationID
AND rprerdfa.DENS = 1 AND rprerdfa.ISOLVER = 3 AND rsolvrdfa.TIME=1.5;
```

Figure 13. DATAFLOW – Analysis of the convergence for a CFD simulation, when it is fixed the fluid density, method of Inexact Newton solver, and the time step.

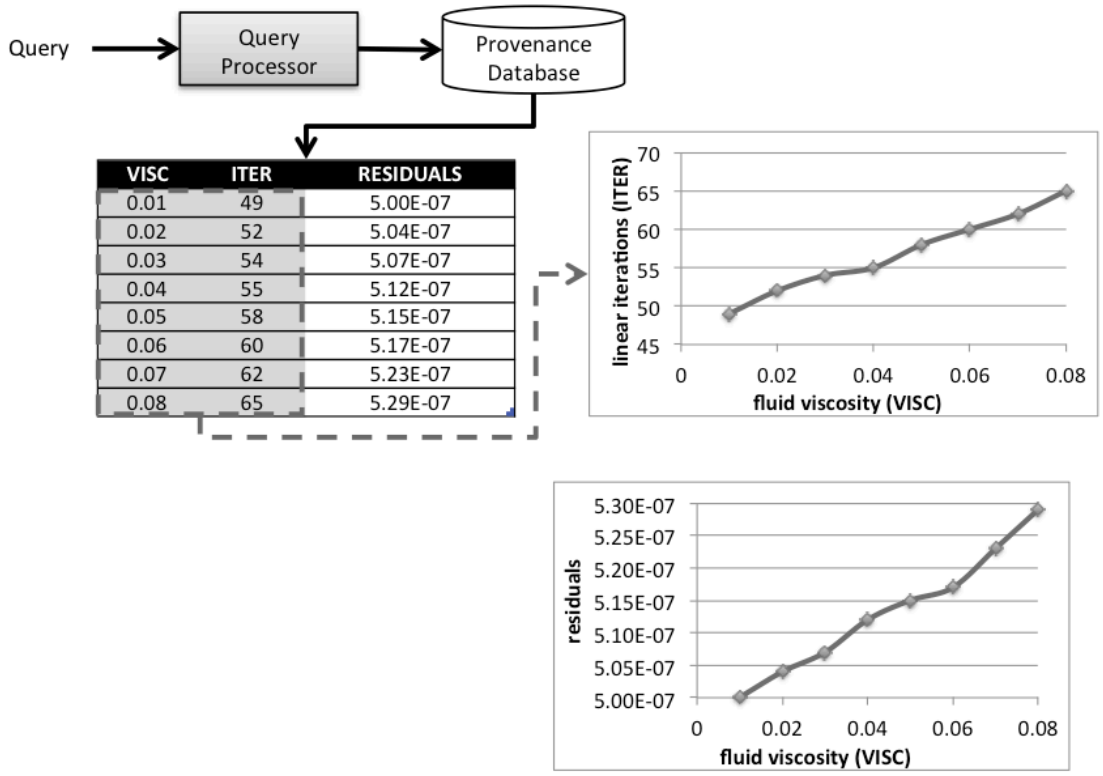


Figure 14. Query results from convergence analysis that aid to analyze the number of linear iterations and residual norms, when users vary the fluid viscosity.

If we consider the workflow versions discussed in this paper, only version *rde* is able to perform those types of query without modifications in our query processor (*i.e.*, users can submit queries directly to the DBMS). However, *idx-bitmap* and *idx-positional* need a specific query processor to be used in queries. This way, it is necessary to access raw data from files using the generated indexes (or index file, if it uses FastBit tool) stored into the DBMS and to join this data with other results obtained from provenance database (for Chiron's provenance database, SQL queries submitted to the relational DBMS). For all CFD workflow versions with index generation, Chiron's DBMS already provides values for density, viscosity, and method of Inexact Newton solver, while the average velocity (from HDF5 files) needs to be accessed by the stored indexes into the output mesh.

Despite the raw data file analysis mechanism introduced in this paper, which supports index generation, we still do not present a formal definition for the query processor using positional and bitmap indexing to support those types of queries. Nonetheless, for the experiments presented in this section, we support these queries using an *ad-hoc* program, which decomposes query to generate and run subqueries. These subqueries isolate operations to be processed in the DBMS used by Chiron's provenance database, retrieve raw data from files using the generated indexes (*e.g.*, query processor from FastBit tool), and join results from the previous steps. When we use bitmap indexing algorithm, we used the query mechanism from FastBit tool to retrieve the attribute values (*i.e.*, raw data elements) based on the indexes stored in the Chiron's provenance database. Furthermore, workflow version without RDFA (*baseline* approach) does not support those queries, since it does not present raw data stored into Chiron's provenance database. Therefore, users are not able to run analytical queries or steer their scientific workflow at runtime.

The third query, named as PERFORMANCE_DATA (see Figure 15), investigates the execution time of the solver in specific data transformations. This query presents similar conditions from query DATAFLOW, since it

considers distinct values for the viscosity, while values are fixed for: density, time step, and a particular setting of the Inexact Newton solver. However, this query combines those raw data to the elapsed time of the CFD solver execution (the elapsed time for executing an activation from the activity *Solver RDFA*). In this case, it is possible to connect data elements from multiple raw data files with metadata associated to the workflow execution, more specifically, the elapsed time for executing a specific activation. This type of query is commonly employed by users to monitor the performance of their CFD solver according to the fluid properties or detect errors. Despite those queries are concentrated in specific data structures, all of them represent an important contribution of our approach: the dataflow management at the logical level (data element flow).

```
SELECT rprerdfa.VISC, a.elapsedTime
FROM RPreRDFA rprerdfa, RSSConfig rconf, RECFDSolver rsolv,
     RSolverRDFA rsolvrdfa, activation a
WHERE rprerdfa.nextActivationID = rconf.previousActivationID
AND rconf.nextActivationID = rsolv.previousActivationID
AND rsolv.nextActivationID = rsolvrdfa.previousActivationID
AND rsolvrdfa.previousActivationID = a.activationID
AND rprerdfa.DENS = 1 AND rprerdfa.ISOLVER = 3;
```

Figure 15. PERFORMANCE_DATA – Performance analysis of CFD solver considering a fixed value for fluid density, and a particular setting of Inexact Newton solver.

Table 3 presents the elapsed time of each query for all workflow versions, *i.e.*, total time to access data from the Chiron's provenance database and an external repository (*e.g.*, usage of the query processor of FastBit tool), and combine those results. According to the query results, we note that *rde* presents a better performance in relation to the workflow versions with indexing algorithms. However, we observe that the time overhead for the query processing using bitmap and positional indexing algorithms, approximately 45.74 minutes in the worst case (query DATAFLOW), is compensated by the time overhead from the execution of programs for extracting raw data from files, generating indexes and ingesting data into the provenance database, which represents approximately 192.84 minutes in experiments INDEXING and INGESTION using fine meshes. Our experiment for query processing considers the average time of 3 executions of those queries, while users commonly perform these queries several times. Considering this scenario, there are some opportunities for adaptive query processing that consider the storage of some metadata or query results with the purpose of improving performance as presented in [8].

Table 3. Elapsed time of query processing.

Workflow version	Query elapsed time (minutes)		
	ELEMENT_FLOW	DATAFLOW	PERFORMANCE_DATA
rde	0.28	4.39	3.88
idx-bitmap	0.51	41.89	41.04
idx-positional	0.56	45.74	44.81

7. Conclusions

Several computer simulations manage a huge volume of raw data within files. This raw data is fundamental for analyzing the evolution of the workflow execution and the results of the experiment. However, analyzing raw data across several files is far from trivial. These resulting data have to be analyzed with their related data elements from multiple raw data files. Solutions based on raw data extraction present a high overhead for loading extracted raw data into an external repository (*e.g.*, a file or a database) to be available for the queries. There is a high cost for parsing and tokenizing the contents from raw data files due to the large size of these files. To avoid this heavy load, some solutions extract raw data on demand based on query submissions [7,8]. However, these existing solutions do not manage data element flow, so users are not able to relate data elements from multiple raw data files.

In our previous work [16], we presented a solution based on the Chiron SWMS with its provenance data support to access and query raw data file and element flow. However, our previous solution still presents a significant raw data load cost and does not provide for direct access to specific raw data file contents. In this paper, we present a dataflow formalism and the definition of operators for raw data extraction and indexing (Raw and RawI). We also separate the raw data analysis services from the SWMS into ARMFUL, so that it can be coupled to other SWMS. Thus, the proposed approach becomes more generic with new algebraic operators and a general definition for dataflows. We define a specialization of W3C PROV so that the provenance database relates raw data

from multiple files to provenance activities and entities. As a proof of concept, we present A-Chiron, an implementation of ARMFUL with Chiron SWMS. The main contributions of ARMFUL are supporting raw data extraction and index generation based on the invocation of external programs (*ad-hoc* programs or third-party tools). Two types of index are currently supported in A-Chiron: a positional indexing algorithm inspired on the RAW system [8], and a bitmap indexing algorithm using the FastBit tool.

Our experiments executed a real scientific workflow with numerical simulation techniques for analyzing the behavior of A-Chiron in fluid dynamics raw data queries. These experiments measured the overhead of A-Chiron and the efficiency obtained by the two proposed indexing techniques. Applying a well-known raw data extraction tool (*e.g.*, ParaView tool) and indexing solutions (*e.g.*, FastBit tool) in A-Chiron, we demonstrate reductions of 52.55% in overhead of data ingestion and 64.95% in the overhead for generating indexes. Runtime queries allowed for the domain users to monitor the convergence of their CFD solver and analyze raw data that assume more advanced data structures (using indexing algorithms), such as meshes. We explore the query capabilities in analyzing raw data from related files. We also show query processing improvements of our indexing techniques and dataflow management. Based on our previous workflow executions with Montage, or in bioinformatics and geophysics we envision that any workflow that can be modeled as a dataflow and present programs, which access raw data from files, can benefit from ARMFUL runtime analytical support.

Our approach based on ARMFUL has two limitations. One is that all references to raw data files are assumed to be valid. We consider that the contents of raw data files will not be modified after the workflow execution although new contents can be added (append-only operation restriction). The second limitation is that the user has to anticipate which domain data should be extracted from the raw data files before the execution starts. It may happen that some data elements from raw data files may turn out to be not relevant for the user analyses. Still, those elements will be extracted or indexed by ARMFUL. To address these limitations, as future work, we intend to provide version control to files and adaptive mechanisms to select domain data to be tracked for dynamic extracting/indexing raw data from files.

Acknowledgements

This work was partially funded by CNPq, CAPES, FAPERJ and Inria (MUSIC project), the European Commission (HPC4E H2020 project) and performed (for P. Valduriez) in the context of the Computational Biology Institute (www.ibc-montpellier.fr). These research results have received funding from the EU H2020 Programme and from MCTI/RNP-Brazil under the HPC4E grant agreement n° 689772. Computer time on Stampede is provided by the Texas Advanced Computer Center (TACC) at University of Texas at Austin. Stampede is supported by the US National Science Foundation under award ACI-1134872.

References

[1] M. Bux, U. Leser, Parallelization in Scientific Workflow Management Systems, CoRR/arXiv:1303.7195, 2013.

[2] S.B. Davidson, J. Freire, Provenance and scientific workflows: challenges and opportunities, in: ACM SIGMOD, ACM, Vancouver, Canada, 2008: pp. 1345–1350. doi:10.1145/1376616.1376772.

[3] J.M. Wozniak, T.G. Armstrong, M. Wilde, D.S. Katz, E. Lusk, I.T. Foster, Swift/T: Large-Scale Application Composition via Distributed-Memory Dataflow Processing, in: CCGrid, IEEE, 2013: pp. 95–102. doi:10.1109/CCGrid.2013.99.

[4] E. Deelman, K. Vahi, G. Juve, M. Rynge, S. Callaghan, P.J. Maechling, R. Mayani, W. Chen, R. Ferreira da Silva, M. Livny, K. Wenger, Pegasus, a workflow management system for science automation, FGCS. 46 (2015) 17–35. doi:10.1016/j.future.2014.10.008.

[5] E. Pennisi, Will Computers Crash Genomics?, Science. 331 (2011) 666–668. doi:10.1126/science.331.6018.666.

[6] A. Romosan, A. Shoshani, K. Wu, V. Markowitz, K. Mavrommatis, Accelerating gene context analysis using bitmaps, in: SSDBM, ACM Press, New York, USA, 2013: p. 1. doi:10.1145/2484838.2484856.

[7] I. Alagiannis, R. Borovica, M. Branco, S. Idreos, A. Ailamaki, NoDB: efficient query execution on raw data files, in: SIGMOD, ACM Press, 2012: p. 241. doi:10.1145/2213836.2213864.

[8] M. Karpathiotakis, M. Branco, I. Alagiannis, A. Ailamaki, Adaptive Query Processing on RAW Data, PVLDB. 7 (2014) 1119–1130.

[9] K. Wu, S. Ahern, E.W. Bethel, J. Chen, H. Childs, E. Cormier-Michel, C. Geddes, J. Gu, H. Hagen, B. Hamann, W. Koegler, J. Lauret, J. Meredith, P. Messmer, E. Otoo, V. Perevoztchikov, A. Poskanzer, Prabhat, O. Rübel, A. Shoshani, A. Sim, K. Stockinger, G. Weber, W.-M. Zhang, FastBit: interactively searching massive data, Journal of Physics: Conference Series. 180 (2009) 012053. doi:10.1088/1742-6596/180/1/012053.

[10] A. Ailamaki, V. Kantere, D. Dash, Managing scientific data, Communications of the ACM. 53 (2010) 68–78. doi:http://doi.acm.org/10.1145/1743546.1743568.
[11] J. Dias, E. Ogasawara, D. Oliveira, F. Porto, A. Coutinho, M. Mattoso, Supporting Dynamic Parameter Sweep in Adaptive and User-Steered Workflow, in: WORKS, ACM, Seattle, WA, USA, 2011: pp. 31–36. doi:10.1145/2110497.2110502.
[12] E. Ogasawara, J. Dias, V. Silva, F. Chirigati, D. Oliveira, F. Porto, P. Valduriez, M. Mattoso, Chiron: A Parallel Engine for Algebraic Scientific Workflows, CCPE. 25 (2013) 2327–2341. doi:10.1002/cpe.3032.
[13] E. Ogasawara, J. Dias, D. Oliveira, F. Porto, P. Valduriez, M. Mattoso, An Algebraic Approach for Data-Centric Scientific Workflows, PVLDB. 4 (2011) 1328–1339.
[14] P. Missier, K. Belhajjame, J. Cheney, The W3C PROV family of specifications for modelling provenance metadata, in: EDBT/ICDT, ACM, New York, NY, USA, 2013: pp. 773–776. doi:10.1145/2452376.2452478.
[15] J. Dias, G. Guerra, F. Rochinha, A.L.G.A. Coutinho, P. Valduriez, M. Mattoso, Data-centric iteration in dynamic workflows, FGCS. 46 (2015) 114–126. doi:10.1016/j.future.2014.10.021.
[16] V. Silva, D. de Oliveira, P. Valduriez, M. Mattoso, Analyzing related raw data files through dataflows, CCPE. 28 (2016) 2528–2545. doi:10.1002/cpe.3616.
[17] G.M. Guerra, S. Zio, J.J. Camata, J. Dias, R.N. Elias, M. Mattoso, P.L. B. Paraizo, A.L. G. A. Coutinho, F.A. Rochinha, Uncertainty quantification in numerical simulation of particle-laden flows, Computational Geosciences. 20 (2016) 265–281. doi:10.1007/s10596-016-9563-6.
[18] R.N. Elias, A.L.G.A. Coutinho, Stabilized edge-based finite element simulation of free-surface flows, International Journal for Numerical Methods in Fluids. 54 (2007) 965–993. doi:10.1002/fld.1475.
[19] Y. Tian, I. Alagiannis, E. Liarou, A. Ailamaki, P. Michiardi, M. Vukolić, DiNoDB: Efficient Large-Scale Raw Data Analytics, in: Data4U, ACM Press, 2014: pp. 1–6. doi:10.1145/2658840.2658841.
[20] B. Dong, S. Byna, K. Wu, SDS: a framework for scientific data services, in: ACM Press, 2013: pp. 27–32. doi:10.1145/2538542.2538563.
[21] N. Fabian, K. Moreland, D. Thompson, A.C. Bauer, P. Marion, B. Geveci, M. Rasquin, K.E. Jansen, The ParaView Coprocessing Library: A scalable, general purpose in situ visualization library, in: LDAV, Oct.: pp. 89–96. doi:10.1109/LDAV.2011.6092322.
[22] Y. Wang, Y. Su, G. Agrawal, Supporting a Light-Weight Data Management Layer over HDF5, in: IEEE, 2013: pp. 335–342. doi:10.1109/CCGrid.2013.9.
[23] M. Bux, J. Brandt, C. Lipka, K. Hakimzadeh, J. Dowling, U. Leser, SAASFEE: scalable scientific workflow execution engine, PVLDB. 8 (2015) 1892–1895.
[24] S. Bowers, T. Mcphillips, S. Riddle, M.K. Anand, B. Ludäscher, Kepler/pPOD: Scientific Workflow and Provenance Support for Assembling the Tree of Life, in: IPAW, Springer-Verlag, Salt Lake City, UT, USA, 2008: pp. 70–77. http://portal.acm.org/citation.cfm?id=1484357 (accessed September 25, 2016).
[25] R. Ikeda, J. Widom, Panda: A System for Provenance and Data, in: IEEE Data Engineering Bulletin, 2010: pp. 42–49.
[26] L. Assuncao, C. Goncalves, J.C. Cunha, Autonomic Activities in the Execution of Scientific Workflows: Evaluation of the AWARD Framework, in: UIC/ATC, Fukuoka, 2012: pp. 423 – 430.
[27] M. Mattoso, J. Dias, K.A.C.S. Ocaña, E. Ogasawara, F. Costa, F. Horta, V. Silva, D. de Oliveira, Dynamic steering of HPC scientific workflows: A survey, FGCS. 46 (2015) 100–113. doi:10.1016/j.future.2014.11.017.
[28] J. Kim, H. Abbasi, L. Chacon, C. Docan, S. Klasky, Q. Liu, N. Podhorszki, A. Shoshani, K. Wu, Parallel in situ indexing for data-intensive computing, in: LDAV, IEEE, 2011: pp. 65–72. doi:10.1109/LDAV.2011.6092319.
[29] J. Chou, K. Wu, Prabhat, FastQuery: A Parallel Indexing System for Scientific Data, in: CLUSTER, IEEE, 2011: pp. 455–464. doi:10.1109/CLUSTER.2011.86.
[30] E.F. Codd, The relational model for database management: version 2, Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1990.
[31] R. Ikeda, A. Das Sarma, J. Widom, Logical provenance in data-oriented workflows?, in: ICDE, IEEE, 2013: pp. 877–888. doi:10.1109/ICDE.2013.6544882.
[32] M. Armbrust, M. Zaharia, T. Das, A. Davidson, A. Ghodsi, A. Or, J. Rosen, I. Stoica, P. Wendell, R. Xin, Scaling spark in the real world: performance and usability, PVLDB. 8 (2015) 1840–1843. doi:10.14778/2824032.2824080.
[33] Y. Amsterdamer, S.B. Davidson, D. Deutch, T. Milo, J. Stoyanovich, V. Tannen, Putting lipstick on pig: enabling database-style workflow provenance, PVLDB. 5 (2011) 346–357.

[34] D. Abramson, B. Bethwaite, C. Enticott, S. Garic, T. Peachey, Parameter Space Exploration Using Scientific Workflows, in: ICCS, Springer Berlin / Heidelberg, 2009: pp. 104–113. http://dx.doi.org/10.1007/978-3-642-01970-8_11.
[35] D. Birsan, On plug-ins and extensible architectures, Queue. 3 (2005) 40–46. doi:10.1145/1053331.1053345.
[36] M.T. Özsu, P. Valduriez, Principles of Distributed Database Systems, 3rd ed., Springer, New York, 2011.
[37] L. Moreau, T.D. Huynh, D. Michaelides, An Online Validator for Provenance: Algorithmic Design, Testing, and API, in: S. Gnesi, A. Rensink (Eds.), Fundamental Approaches to Software Engineering, Springer Berlin Heidelberg, Berlin, Heidelberg, 2014: pp. 291–305. http://link.springer.com/10.1007/978-3-642-54804-8_20 (accessed August 25, 2016).
[38] F. Horta, V. Silva, F. Costa, D. de Oliveira, K. Ocaña, E. Ogasawara, J. Dias, M. Mattoso, Provenance Traces from Chiron Parallel Workflow Engine, in: EDBT/ICDT, ACM, New York, NY, USA, 2013: pp. 337–338. doi:10.1145/2457317.2457379.
[39] P. Boncz, T. Grust, M. van Keulen, S. Manegold, J. Rittinger, J. Teubner, MonetDB/XQuery: a fast XQuery processor powered by a relational engine, in: SIGMOD, ACM, New York, NY, USA, 2006: pp. 479–490. doi:10.1145/1142473.1142527.
[40] K.A.C.S. Ocaña, D. Oliveira, E. Ogasawara, A.M.R. Dávila, A.A.B. Lima, M. Mattoso, SciPhy: A Cloud-Based Workflow for Phylogenetic Analysis of Drug Targets in Protozoan Genomes, in: O. Norberto de Souza, G.P. Telles, M. Palakal (Eds.), Advances in Bioinformatics and Computational Biology (BSB), Springer, Berlin, Heidelberg, 2011: pp. 66–70. http://www.springerlink.com/index/10.1007/978-3-642-22825-4_9 (accessed August 25, 2011).
[41] D.C. Lo, K. Murugesan, D.L. Young, Numerical solution of three-dimensional velocity-vorticity Navier-Stokes equations by finite difference method, International Journal for Numerical Methods in Fluids. 47 (2005) 1469–1487. doi:10.1002/fld.822.
[42] I. Raicu, I.T. Foster, Yong Zhao, Many-task computing for grids and supercomputers, in: MTAGS, Austin, Texas, USA, 2008: pp. 1–11.