DfAnalyzer: Runtime Dataflow Analysis Tool for Computational Science and Engineering Applications

DfAnalyzer

Vítor Silva[*] / COPPE, Federal University of Rio de Janeiro, Brazil / silva@cos.ufrj.br
Vinícius Campos / COPPE, Federal University of Rio de Janeiro, Brazil / vinicius.s.campos@poli.ufrj.br
Thaylon Guedes / Fluminense Federal University, Brazil / thaylongs@id.uff.br
José Camata / Federal University of Juiz de Fora, Brazil / camata@ice.ufjf.br
Daniel de Oliveira / Fluminense Federal University, Brazil / danielcmo@ic.uff.br
Alvaro L. G. A. Coutinho / COPPE, Federal University of Rio de Janeiro, Brazil / alvaro@nacad.ufrj.br
Patrick Valduriez / Inria, University of Montpellier, CNRS, LIRMM, France / patrick.valduriez@inria.fr
Marta Mattoso / COPPE, Federal University of Rio de Janeiro, Brazil / marta@cos.ufrj.br

Abstract.
DfAnalyzer is a tool for monitoring, debugging, and analyzing dataflows generated by Computational Science and Engineering (CSE) applications. It collects strategic raw data, registering provenance data, and enabling query processing, all asynchronously and at runtime. DfAnalyzer provides lightweight dataflow components to be invoked by CSE applications using High-Performance Computing (HPC), in the same way computational scientists plug HPC (e.g., PETSc) and visualization (e.g., ParaView) libraries. We show DfAnalyzer's main functionalities and how to analyze dataflows in CSE applications at runtime. The performance evaluation of CSE executions for a complex multiphysics application shows that DfAnalyzer has negligible time overhead on the total elapsed time.

Keywords:
Computational Science and Engineering (CSE); dataflow; provenance; computational applications; data analysis

1. *Motivation and Significance*

Computational Science and Engineering (CSE) applications rely on complex mathematical models that solve problems typically requiring High-Performance Computing (HPC) [1]. They can be found in biology, chemistry, geology, several engineering areas, etc. They have the exploratory nature of scientific applications, with large-scale executions that can last for a long time, even using HPC. The software ecosystem for developing these applications involves much more than writing scripts or invoking a chain of legacy scientific codes. Computational scientists develop simulation codes that invoke components of CSE frameworks and libraries. For example, components are invoked to provide: (i) support for Partial Differential Equation (PDE) discretization methods in libraries like libMesh, FEniCS, MOOSE, deal.II, GREENS, OpenFOAM; (ii) algorithms for solving numerical problems with parallel computations, in libraries like PETSc, LAPACK, SLEPc; (iii) runtime visualization, like ParaView Catalyst, VisIt, SENSEI; (iv) parallel graph partitioning, like ParMetis, Scotch; and (v) I/O data management like ADIOS.

Several parameters have to be set to invoke library components from these frameworks. Often, these parameters are difficult to preset, and thus need monitoring and debugging capabilities for runtime fine-tuning. Computational scientists often use in situ visualization techniques to provide information to help control simulations [2]. By observing a specific pattern, an experienced interpreter can infer that something is not going well in the simulation, deciding to stop it or change parameters, preferably at runtime, resuming or adapting the simulation. However, to do that, the visualization should be complemented with information regarding the evolution of quantities of interest

[*] Vitor Silva is now at Snap Inc., USA

(QoI), such as residual norms, number of linear and nonlinear iterations, often within a specific time window, not just the current values. To obtain this information, even the experienced interpreter has difficulty in identifying the files related to the time window, opening and parsing them to obtain specific values, filtering, aggregating and tracking their evolution.

DfAnalyzer addresses the problem of generating simulation data for runtime analysis by integrating in situ visualization data with raw data capture in large-scale parallel CSE applications. One of the challenges is to track data evolution because raw data files are distributed and have implicit data relationships in their file contents. DfAnalyzer identifies these data relationships as a dataflow [3]. This dataflow is the representation of data resulting from the composition of the CSE application programs that execute data transformations. Each data transformation consumes data from one (or more) dataset(s) as input and produces data in one (or more) dataset(s) as output. Each dataset is composed of a set of data elements. Two data transformations can present a data dependency with relation to a dataset, when the data elements are produced by one data transformation and consumed by another. Another challenge is to preserve the autonomy of the CSE execution control code and not compete with resources allocated for running the CSE applications. The main performance challenge is related to capturing data suitable for runtime analysis that does not interfere with CSE execution control and data management. This data capture has to address data modeling, representation, loading, storage, monitoring and steering, while avoiding replicating file contents and keeping the autonomy of binary files like in HDF5 or other formats.

DfAnalyzer is a lightweight tool for runtime collection, management, monitoring, and analysis of distributed provenance data generated by CSE applications in HPC environments. It uses provenance data [4] to help registering parameter choices and associating them with intermediate data and results. Provenance enables the traceability and reproducibility of CSE applications and improves data analysis and adaptation at runtime [5]. The main contribution of DfAnalyzer is to provide dataflow extraction [6], with negligible overhead, as an integrated view of QoI with generation traces based on the W3C PROV standard provenance data representation. DfAnalyzer follows a PROV-compliant data model, for representing relationships between datasets manipulated by computational models, which is agnostic concerning the scientific application domain. DfAnalyzer innovates by extracting and relating raw data (e.g., QoI) from heterogeneous distributed files at runtime. It accesses strategic domain data associated to these files using in situ and in transit raw data extraction approaches. It stores dataflow provenance associated to extracted raw data all in a columnar database, which acts as a global map of the CSE application raw data while the application executes. This dataflow map allows for monitoring queries like *what is the average error estimate calculated in all iterations so far*. User steering actions with DfAnalyzer are discussed in [7] with a fluid dynamics application built with libMesh [8], in a Python script [9] with FEnICS [10], and in [3] with Spark in a simple business application.

DfAnalyzer is used with the following methodology [9]. Initially the user (CSE application developer) interacts with a database expert to help on the modeling of the raw data that should be extracted. The user identifies data items to be tracked and how they relate to other data along their lineage within the CSE code. The database specialist models the data transformation chain using W3C PROV activities and entities with extensions for the raw data items, QoI and parameters. The result of this data modeling is then mapped to a provenance database. The participation of the user in this data modeling saves a lot of time during data analyses and helps on query formulations. The user selectively chooses only application data of interest to be registered, providing a coarse-grain relevant provenance data and selected raw data. Then, DfAnalyzer library calls are inserted in the CSE application as *input, output, task* and *output* followed by an *extracted data* call. DfAnalyzer has a set of RESTful services (and libraries on C++, Python, and Java) to help plugging the calls into the CSE applications. The components that are invoked capture data asynchronously during CSE application execution. They send all insert/update requests to a columnar database system that runs in computing nodes that are different from the CSE application.

There are three main approaches to provide runtime data analysis for CSE applications. Unlike DfAnalyzer, they all fail at either preserving the autonomy of the CSE execution control code or competing with resources allocated for running the CSE applications. The first approach is the class of HPC workflow systems [11]. Workflow systems collect provenance data at runtime, but they must be in control of the execution flow and data transfer. This is in conflict with CSE applications, since they call typical libraries for solving numerical problems with parallel execution control, often with parallel visualizations. Provenance data management systems like noWorkflow [12] and PROV-Template [13] do preserve the autonomy of CSE execution control, but they compete with computing resources, thus producing a high overhead and do not run in HPC. The third approach provides for data analysis independent from the CSE application, *i.e.* unlike workflow systems, the same CSE code can run with or without the data analyzer. This is the most similar approach to DfAnalyzer, with solutions like Spade [14] and ADIOS [15]. However, they have limited data analysis, because they do not help on extracting raw data or providing a dataflow view with provenance data. Approaches like FastBit [16] and PostgresRaw [17] are helpful for accessing and extracting raw data from heterogeneous files, but they operate in post-processing mode and thus are not aware of the implicit dataflow between raw data files, being complementary to DfAnalyzer. We know of no other solution for dataflow analysis at runtime. In [18], we present the main open issues for the exploratory runtime scientific data analysis scenarios. DfAnalyzer's user steering support contributes to the process of scientific discovery and explainability by automatically adding provenance, context and relationships to scientific data native formats.

2. ***Software Description***

DfAnalyzer assumes that the CSE programs are white or gray boxes, and works as a data profiling tool, similarly to code profilers. DfA-lib is the library invoked by DfAnalyzer calls. As the CSE application executes, DfA-lib extracts raw data and captures provenance data, at key points of the application, to map the dataflows. DfAnalyzer has a component-based architecture [6], which allows registering plug-ins for raw data extraction and indexing. As in software engineering, a plug-in refers to the extension of a component behavior or actions so that users can reuse these plug-ins to minimize efforts of developing CSE application data analyses.

2.1. Software Architecture

The DfAnalyzer architecture has three layers (Figure 1):

- *Storage*, which accesses its database and raw data from files manipulated by the application;
- *Dataflow*, which presents the main components for capturing provenance and raw data, and for registering the dataflow implicit in the application; and
- *Data Analysis*, which provides graphical interfaces to ease dataflow monitoring and querying.

The *Storage* layer has the database containing selected raw data and the id of binary files consumed and generated by the application as a provenance dataflow. Execution data related to provenance data are also captured. The database stores both raw data extracted from files and the references to these files, which are registered as pointers (*e.g.*, URIs or file paths).

The *Dataflow* layer has four components: *Provenance Data Extractor* (PDE), *Raw Data Extractor* (RDE), *Raw Data Indexer* (RDI), and *Query Interface* (QI). PDE captures provenance and domain data, RDE invokes *ad-hoc* programs to extract raw data from files, and RDI applies indexing techniques to minimize the data loading cost in DfAnalyzer's database and improve the performance of query processing on scientific data. QI translates query requests generated by the *Query Dashboard*, according to the user interactions, to a database query specification. This query runs on DfAnalyzer's database system (currently MonetDB), accessing when necessary, the file contents.

The *Data Analysis* layer works as a dashboard with *Dataflow Viewer* (DfViewer) and *Query Dashboard* (QD). QD provides raw data analysis interacting with QI and presenting the query results in text tables. DfViewer presents a graphical dataflow specification based on the execution of the CSE application. This dashboard shows data transformations, datasets, attributes, and data dependencies of each dataflow specification stored in the database.

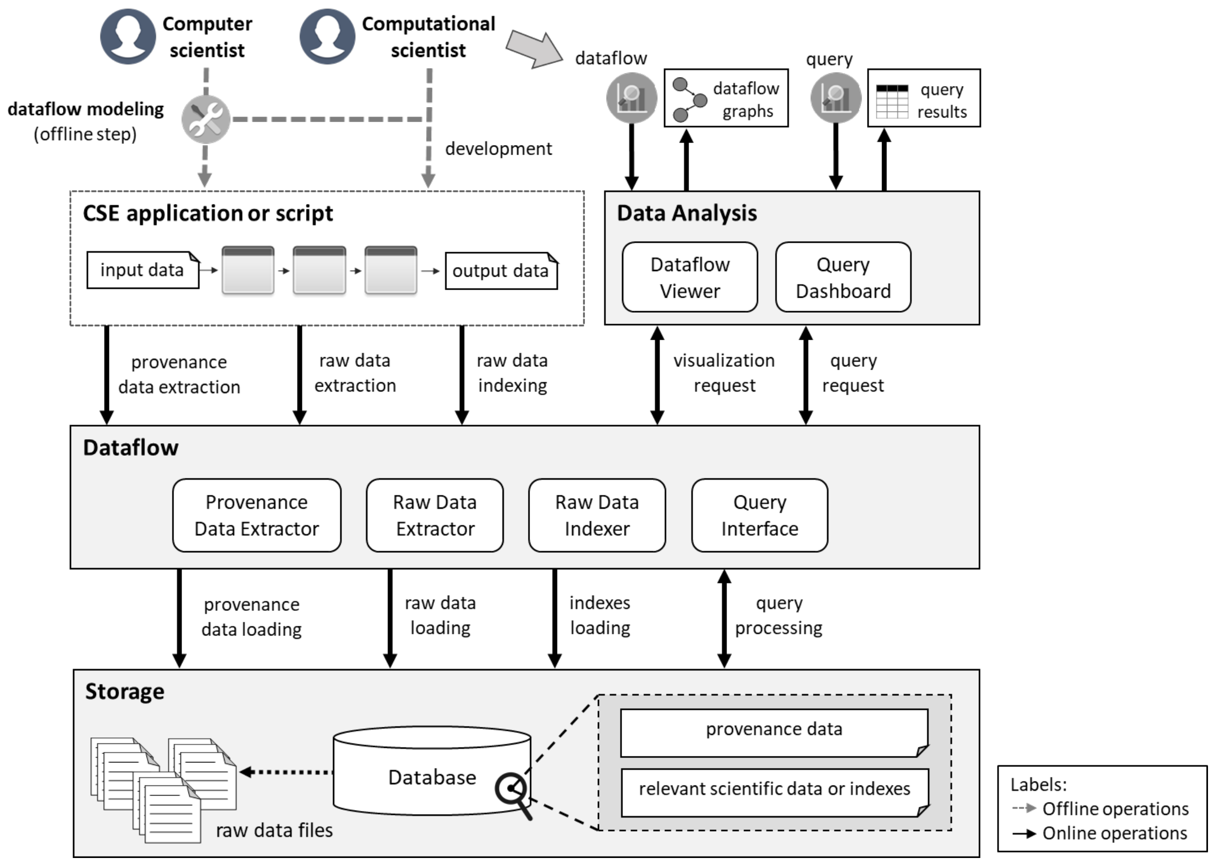


Figure 1. DfAnalyzer architecture.

The gray and black dotted arrows in Figure 1 represent *offline* and *online operations*. Offline operations correspond to the dataflow modeling steps based on the inclusion of DfAnalyzer library calls specifying points of provenance capture and raw data extraction/indexing in CSE applications. Thus, these operations take place before the execution starts. Meanwhile, online operations are associated to the provenance capture, raw data extraction/indexing, and query processing executed by DfAnalyzer's components at runtime.

2.2. Software Functionalities

DfAnalyzer has two components for raw data extraction and indexing, and a RESTful web application that receives different kinds of HTTP requests from CSE applications or web browsers. DfAnalyzer implements plug-ins on the RDE and RDI components for raw data extraction and indexing, respectively. Furthermore, HTTP requests trigger RESTful services for capturing provenance data and running queries, which involve the following components of DfAnalyzer: DfViewer, QD, PDE, and QI. Figure 2 presents an overview of this design and the following subsections describe the implementation of each component.

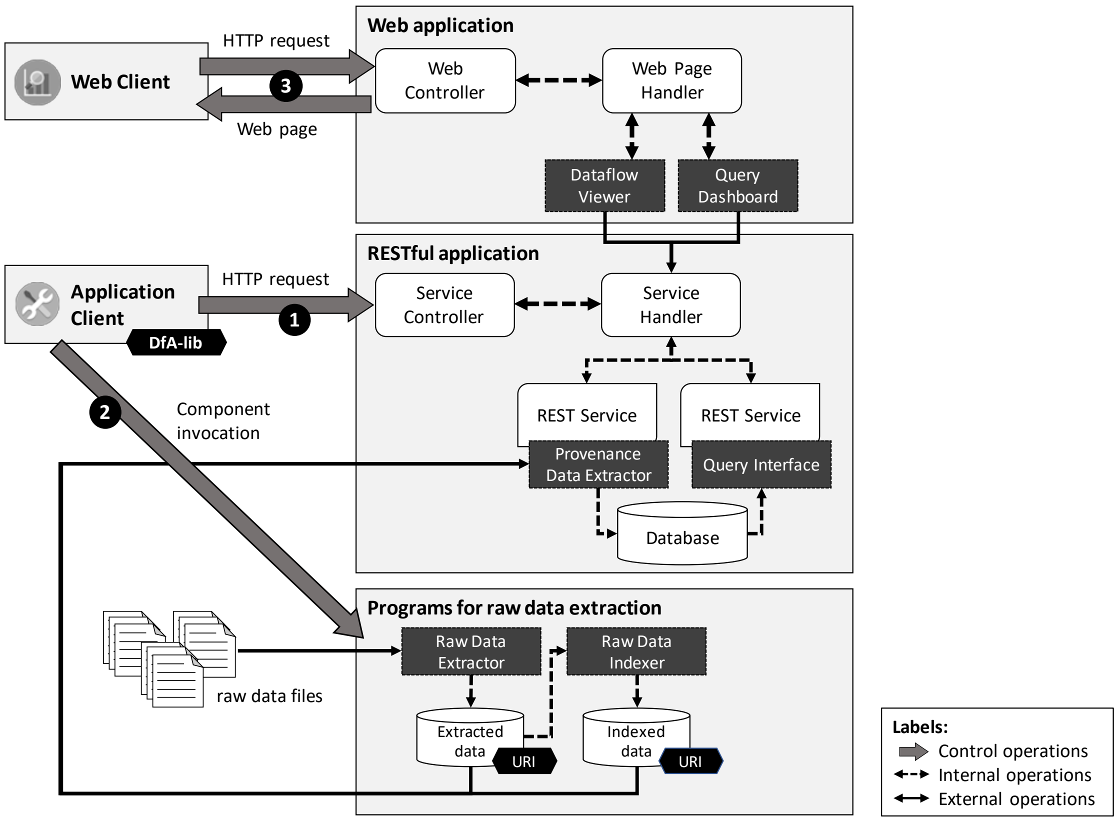


Figure 2. Design of DfAnalyzer's implementation considering RESTful services and raw data extraction/indexing components.

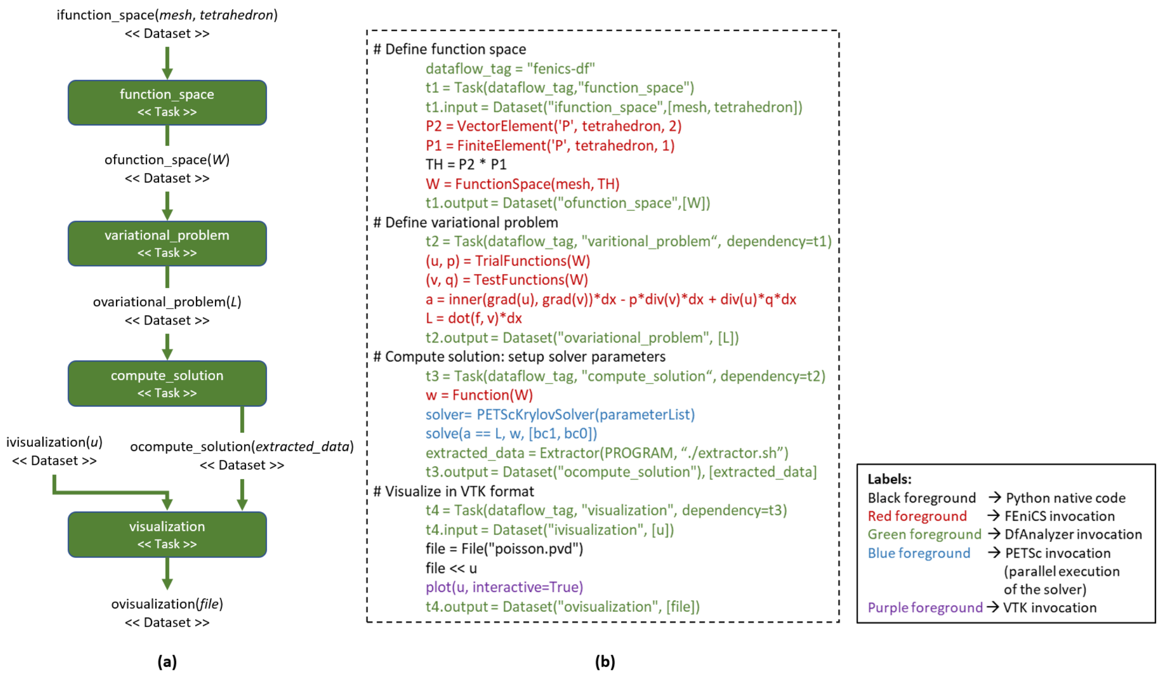
RDE and RDI components are responsible to run *ad-hoc* programs and third-party tools for accessing, parsing, tokenizing, extracting, and, occasionally, indexing relevant contents from data sources, such as raw data files. Each algorithm or solution for raw data extraction or indexing is implemented respectively as a plug-in in RDE or RDI. RDE presents the *PROGRAM* plug-in that accesses the relevant contents of raw data files by invoking *ad-hoc* programs developed by the user, while the *CSV* plug-in extracts data from CSV files.

DfA-lib calls, inside the CSE application code, send HTTP requests to the *Service Controller* with a POST method using DfAnalyzer's RESTful web application. There are two body types in these HTTP requests, with prospective and retrospective provenance data [19]. A request body with prospective provenance data contains information about the dataflow structure of an application and, consequently, it represents a mapping to the dataflow concepts [18]. By contrast, a request body with retrospective provenance data corresponds to the execution and raw data generated/extracted by the application at runtime.

More specifically, once the *Service* Controller receives HTTP requests for provenance capture, it translates them and creates a service for each request. In this case, a service is created for provenance and raw data loading. Then, it is queued in the *Service Handler* and waits until all previous services are processed by this handler. To run the analytical queries, a service is created by the *Service Handler* for submitting and running queries on the database using the QI component.

3. *Illustrative Examples*

Figure 3 shows a fragment of the FEniCS Python code for solving the Cahn-Hilliard equation, a mathematical model from material science. The Cahn-Hilliard equation leads to a prototype of a transient nonlinear multi-physics code. Several parameters have to be set to invoke these components, which are very difficult to preset and need monitoring for runtime fine-tuning. Provenance data can help in registering parameter choices with the results. Thus, associating them as a dataflow, as in Figure 3(a), can improve both runtime data analysis and fine-tuning. The corresponding dataflow is extracted by inserting DfA-lib calls on the Python script, as shown in Figure 3(b).



```
# Define function space
        dataflow_tag = "fenics-df"
        t1 = Task(dataflow_tag,"function_space")
        t1.input = Dataset("ifunction_space",[mesh, tetrahedron])
        P2 = VectorElement('P', tetrahedron, 2)
        P1 = FiniteElement('P', tetrahedron, 1)
        TH = P2 * P1
        W = FunctionSpace(mesh, TH)
        t1.output = Dataset("ofunction_space",[W])
# Define variational problem
        t2 = Task(dataflow_tag, "varitional_problem", dependency=t1)
        (u, p) = TrialFunctions(W)
        (v, q) = TestFunctions(W)
        a = inner(grad(u), grad(v))*dx - p*div(v)*dx + div(u)*q*dx
        L = dot(f, v)*dx
        t2.output = Dataset("ovariational_problem", [L])
# Compute solution: setup solver parameters
        t3 = Task(dataflow_tag, "compute_solution", dependency=t2)
        w = Function(W)
        solver= PETScKrylovSolver(parameterList)
        solve(a == L, w, [bc1, bc0])
        extracted_data = Extractor(PROGRAM, "./extractor.sh")
        t3.output = Dataset("ocompute_solution"), [extracted_data]
# Visualize in VTK format
        t4 = Task(dataflow_tag, "visualization", dependency=t3)
        t4.input = Dataset("ivisualization", [u])
        file = File("poisson.pvd")
        file << u
        plot(u, interactive=True)
        t4.output = Dataset("ovisualization", [file])
```

Labels:

Black foreground	→ Python native code
Red foreground	→ FEniCS invocation
Green foreground	→ DfAnalyzer invocation
Blue foreground	→ PETSc invocation (parallel execution of the solver)
Purple foreground	→ VTK invocation

Figure 3. Finite element solution of Cahn-Hilliard equation with FEniCS- (a) dataflow and (b) code with library calls in colors.

The DfA-lib calls (input, output, task, and extracted data) register, as a dataflow, mesh information, variational model, solver parameters (as solver tolerances, number of iterations and residual norms) and visualization. Relating these data allows for filtering and directly accessing them. The dataflow is extracted and inserted in the DBMS while the user can steer the execution with queries like "for all converged steps, show the number of iterations, the corresponding residual norms, and the output visualization". With this query result, the user can fine-tune in runtime, the nonlinear solver tolerance to speed-up the simulation without compromising its accuracy.

In [7], DfAnalyzer is used in a real CSE application, built with libMesh, for the finite element parallel adaptive mesh refinement/coarsening simulation of turbidity currents. DfAnalyzer gathered QoI (residual norms, number of linear and nonlinear iterations); extracted data in memory using ParaView Catalyst, such as velocity, pressure and sediment concentration; gathered provenance data; and relating these simulation data from different files in its database. Users observed intermediate data at different points in time, complemented by visualization files to see if the simulation had already all relevant results and could resume or needed more time. If the decision is to continue, they need to reset the maximum time interval, which is also registered in the database. Examples of queries submitted during this execution are: which are the sediment deposits in a region delimited by x in range [9, 13.5]; what is the elapsed time spent in the different stages of the iterations; show for each time step and for each of its nonlinear/linear iterations, the residual norm and its convergence status (Table 1). Based on these monitoring and debugging analyses, the user was confident to reset some solver parameters without interrupting the simulation run, which reduced the elapsed time in days. Without the dataflows of DfAnalyzer, they would need to find and parse the related files to extract, relate and analyze raw data, possibly after a failed execution, wasting valuable HPC resources. Our GitLab repository[1] documents DfAnalyzer's components, shows DfA-lib installation steps and provides a Docker image with these two CSE examples[2] presented in this section.

[1] https://gitlab.com/ssvitor/dataflow_analyzer/-/tree/master/library

[2] https://gitlab.com/ssvitor/dataflow_analyzer/-/tree/master/applications

The adoption of a provenance system in CSE depends on how much execution overhead it adds to the application. In DfAnalyzer, this execution overhead depends on the data identified in the simulation code that needs to be tracked. That is, which input and output data values, for each data transformation, should be extracted and registered to be monitored during the execution. Therefore, the added overhead can be defined as the sum of time costs for extracting/indexing these data from raw data files and monitoring the provenance of all data defined by the user. When using DfAnalyzer with several real CSE applications in large scale, the added overhead has been always below 1% [7,20,21]. In small scale use cases [3,9], that run for minutes in desktops, this overhead becomes <3%, since data is generated very fast and does not benefit from DfAnalyzer's asynchronous data extraction.

Table 1. Query results for numerical analysis with libMesh to detect possible misbehavior of nonlinear and linear solvers.

time step	Flow		Transport		Visualization
	Linear residual	Nonlinear residual	Linear residual	Nonlinear residual	
1000	0.0168372	0.0053668	0.000226965	0.0000000	/viz/img_1000.png
2000	0.0643741	1.75431e-06	0.000109237	0.00186711	/viz/img_2000.png
3000	0.1270304	0.0027324	0.000230409	3.265e-06	/viz/img_3000.png
			...		

4. *Impact*

The Interoperable Design of Extreme-scale Application Software (IDEAS) [22] is a family of projects, involving several institutions in the USA, concerned with the complexity of developing software for CSE applications. IDEAS aims at "enabling a fundamentally different attitude to creating and supporting CSE applications" with desirable features like provenance and reproducibility [5]. libMesh, FEniCS, and ParaView are softwares considered in the IDEAS project. DfAnalyzer models CSE applications as activities and dataflows represented as provenance data in the W3C standard. This generic standard data model helps the user in identifying data and activities of the CSE dataflow regardless of the application domain.

The DfAnalyzer approach defines a specific software layer for data analysis, in the same way visualization libraries provide specific components for visualization. Encapsulating all data analysis support in one specific library/component, as DfA-lib, makes the application autonomous and data extraction for analyses can always be switched off. The overhead on raw data capture depends on how much data is to be extracted but an in-situ approach avoids data movements and using indexes reduces this overhead. DfAnalyzer contributes to improve developing software for CSE applications because the developer does not have to write logging code for each data analysis. The same data model can be reused, and the databases associated to all the executions can be further analyzed using AI tools to improve CSE parameter settings and finding correlations among distributed data. The DfA-lib can complement data systems in other layers like IO libraries, burst buffers and visualization tools.

5. *Conclusion*

Encapsulating all data analytics support in one specific library/component makes the application autonomous and data analyses can always be switched off. There are several advantages of this data analysis approach. First, simulation data is preserved in their format and not fully replicated in the DBMS. Second, data is related among different files while it is being generated, which might be cumbersome after the simulation ends, as in post-processing approaches. Third, the history of data generation is registered for further analysis or reproduction through provenance, following W3C PROV. Fourth, efficient data management techniques from column-oriented relational DBMS can be used at runtime. Consequently, DfAnalyzer provides a provenance database enriched with quantities of interest (*i.e.*, domain data) in CSE applications that can be queried at runtime and offline.

Acknowledgements
This work has been partially funded by Inria (SciDISC and HPDaSc associated teams), CAPES, CNPq, FAPERJ, and the HPC4E project (EU H2020 and MCTI/RNP-Brazil). The HPC Center at COPPE/ Federal University of Rio de Janeiro has provided computing and storage resources.

References

[1] U. Rüde, K. Willcox, L.C. McInnes, H.D. Sterck, G. Biros, H.-J. Bungartz, J. Corones, E. Cramer, J. Crowley, O. Ghattas, M. Gunzburger, M. Hanke, R.J. Harrison, M.A. Heroux, J. Hesthaven, P.K. Jimack, C. Johnson, K.E. Jordan, D.E. Keyes, R.H. Krause, V. Kumar, S. Mayer, J. Meza, K.M. Mørken, J.T. Oden, L.R. Petzold, P. Raghavan, S.M. Shontz, A.E. Trefethen, P.R. Turner, V.V. Voevodin, B.I. Wohlmuth, C.S. Woodward, Research and Education in Computational Science and Engineering, CoRR. abs/1610.02608 (2016). http://arxiv.org/abs/1610.02608.

[2] A.C. Bauer, H. Abbasi, J. Ahrens, H. Childs, B. Geveci, S. Klasky, K. Moreland, P. O'Leary, V. Vishwanath, B. Whitlock, E.W. Bethel, In Situ Methods, Infrastructures, and Applications on High Performance Computing Platforms, Computer Graphics Forum. 35 (2016) 577–597. https://doi.org/10.1111/cgf.12930.

[3] V. Silva, D. De Oliveira, P. Valduriez, M. Mattoso, DfAnalyzer: Runtime Dataflow Analysis of Scientific Applications using Provenance, in: International Conference on Very Large Data Bases, Rio de Janeiro, Brazil, 2018.

[4] J. Freire, D. Koop, E. Santos, C.T. Silva, Provenance for Computational Tasks: A Survey, Computing in Science and Engineering. 10 (2008) 11–21. https://doi.org/10.1109/MCSE.2008.79.

[5] D. Bernholdt, A. Dubey, M. Heroux, A. Klinvex, L.C. McInnes, Improving Reproducibility Through Better Software Practices, (2017).

[6] V. Silva, J. Leite, J. Camata, D. Oliveira, A.L.G.A. Coutinho, P. Valduriez, M. Mattoso, Raw Data Queries during Data-intensive Parallel Workflow Execution, Special Issue on Workflows for Data-Driven Research in the Future Generation Computer Systems Journal. (2017).

[7] J.J. Camata, V. Silva, P. Valduriez, M. Mattoso, A.L.G.A. Coutinho, In situ visualization and data analysis for turbidity currents simulation, Computers & Geosciences. 110 (2018) 23–31. https://doi.org/10.1016/j.cageo.2017.09.013.

[8] B.S. Kirk, J.W. Peterson, R.H. Stogner, G.F. Carey, libMesh : a C++ library for parallel adaptive mesh refinement/coarsening simulations, Engineering with Computers. 22 (2006) 237–254. https://doi.org/10.1007/s00366-006-0049-3.

[9] V. Silva, R. Souza, J. Camata, D. de Oliveira, A.L.G.A. Coutinho, P. Valduriez, M. Mattoso, Capturing Provenance for Runtime Data Analysis in Computational Science and Engineering Applications, in: International Provenance and Annotation Workshop (IPAW), London, UK, 2018.

[10] M. Alnæs, J. Blechta, J. Hake, A. Johansson, B. Kehlet, A. Logg, C. Richardson, J. Ring, M.E. Rognes, G.N. Wells, Archive Of Numerical Software, Archive of Numerical Software: The FEniCS Project Version 1.5, University Library Heidelberg, 2015. https://doi.org/10.11588/ans.2015.100.20553.

[11] J. Liu, E. Pacitti, P. Valduriez, M. Mattoso, A Survey of Data-Intensive Scientific Workflow Management, Journal of Grid Computing. 13 (2015) 457–493. https://doi.org/10.1007/s10723-015-9329-8.

[12] J.F. Pimentel, L. Murta, V. Braganholo, J. Freire, noWorkflow: a tool for collecting, analyzing, and managing provenance from python scripts, Proceedings of the VLDB Endowment. 10 (2017) 1841–1844. https://doi.org/10.14778/3137765.3137789.

[13] L. Moreau, B.V. Batlajery, T.D. Huynh, D. Michaelides, H. Packer, A Templating System to Generate Provenance, IEEE Transactions on Software Engineering. 44 (2018) 103–121. https://doi.org/10.1109/TSE.2017.2659745.

[14] Scaling SPADE to "Big Provenance," in: 8th USENIX Workshop on the Theory and Practice of Provenance (TaPP 16), USENIX Association, Washington, D.C., 2016. https://www.usenix.org/conference/tapp16/workshop-program/presentation/gehani.

[15] Q. Liu, J. Logan, Y. Tian, H. Abbasi, N. Podhorszki, J.Y. Choi, S. Klasky, R. Tchoua, J. Lofstead, R. Oldfield, M. Parashar, N. Samatova, K. Schwan, A. Shoshani, M. Wolf, K. Wu, W. Yu, Hello ADIOS: the challenges and lessons of developing leadership class I/O frameworks, Concurrency and Computation: Practice and Experience. 26 (2014) 1453–1473. https://doi.org/10.1002/cpe.3125.

[16] 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. https://doi.org/10.1088/1742-6596/180/1/012053.

[17] I. Alagiannis, R. Borovica-Gajic, M. Branco, S. Idreos, A. Ailamaki, NoDB: efficient query execution on raw data files, Communications of the ACM. 58 (2015) 112–121. https://doi.org/10.1145/2830508.

[18] V. Silva, D. de Oliveira, P. Valduriez, M. Mattoso, Analyzing related raw data files through dataflows, CCPE. 28 (2016) 2528–2545. https://doi.org/10.1002/cpe.3616.

[19] S.B. Davidson, J. Freire, Provenance and Scientific Workflows: Challenges and Opportunities, in: ACM SIGMOD, ACM, New York, NY, USA, 2008: pp. 1345–1350. https://doi.org/10.1145/1376616.1376772.

[20] V. Silva, J. Camata, D. de Oliveira, A.L.G.A. Coutinho, P. Valduriez, M. Mattoso, In Situ Data Steering on Sedimentation Simulation with Provenance Data, in: Poster Session of Supercomputing Conference, 2016.

[21] R. Souza, V. Silva, J. Camata, A. Coutinho, V. Valduriez, M. Mattoso, Keeping track of user steering actions in dynamic workflows, Future Generation Computer Systems. 99 (2019) 624–643.

[22] IDEAS (Interoperable Design of Extreme-scale Application Software), (n.d.). https://ideas-productivity.org.

Required Metadata

Current code version

Table 1 – Code metadata

Nr	**Code metadata description**	**Information**
C1	Current code version	1.0
C2	Permanent link to code/repository used of this code version	https://gitlab.com/ssvitor/dataflow_analyzer
C3	Legal Code License	MIT License (MIT)
C4	Code versioning system used	Git
C5	Software code languages, tools, and services used	Java, Spring Boot, Apache Maven
C6	Compilation requirements, operating environments & dependencies	Java Development Kit (JDK), MonetDB, FastBit
C7	If available Link to developer documentation/manual	https://gitlab.com/ssvitor/dataflow_analyzer
C8	Support email for questions	marta@cos.ufrj.br

Current executable software version

Table 2 – Software metadata

Nr	**(Executable) software metadata description**	**Information**
S1	Current software version	1.0
S2	Permanent link to executables of this version	https://gitlab.com/ssvitor/dataflow_analyzer/tree/master/applications/dfanalyzer/dfa
S3	Legal Software License	MIT License (MIT)
S4	Computing platforms/Operating Systems	Unix Systems
S5	Installation requirements & dependencies	Java Development Kit (JDK), MonetDB, FastBit
S6	If available, link to user manual - if formally published include a reference to the publication in the reference list	https://gitlab.com/ssvitor/dataflow_analyzer
S7	Support email for questions	marta@cos.ufrj.br