Experiencing DfAnalyzer for Runtime Analysis of Phylogenomic Dataflows[*]

Luiz Gustavo Dias[1][0000−0002−2178−5527], Marta Mattoso[2][0000−0002−0870−3371], Bruno Lopes[1][0000−0003−1204−0176], and Daniel de Oliveira[1][0000−0001−9346−7651]

[1] Fluminense Federal University, IC/UFF
`lgdias@id.uff.br, {bruno,danielcmo}@ic.uff.br`
[2] Federal University of Rio de Janeiro, COPPE/UFRJ
`marta@cos.ufrj.br`

Abstract. Phylogenomic experiments provide the basis for evolutionary biology inferences. They are data- and CPU-intensive by nature and aim at producing phylogenomic trees based on an input dataset of protein sequences of genomes. These experiments can be modeled as scientific workflows. Although workflows can be efficiently managed by Workflow Management Systems (WfMS), they are not often used by bioinformaticians, which traditionally use scripts to implement their workflows. However, collecting provenance from scripts is a challenging task. In this paper, we specialize the DfAnalyzer tool for the phylogenomics domain. DfAnalyzer enables capturing, monitoring, debugging, and analysing dataflows while being generated by the script. Additionally, it can be invoked from scripts, in the same way bioinformaticians already import libraries in their code. The proposed approach captures strategic domain data, registering provenance and telemetry (performance) data to enable queries at runtime. Another advantage of specializing DfAnalyzer in the context of Phylogenomic experiments is the capability of capturing data from experiments that execute either locally or in HPC environments. We evaluated the proposed specialization of DfAnalyzer using the SciPhylomics workflow and the proposed approach showed relevant telemetry scenarios and rich data analyses.

Keywords: Scientific Workflow · Provenance · Dataflow Analysis.

1 Introduction

Over the past years, several categories of experiments in the bioinformatics domain became more and more dependent on complex computational simulations. One example are the Phylogenomic analyses that provide the basis for evolutionary biology inferences, and they have been fostered by several technologies (*e.g.*, DNA sequencing methods [10] and novel mathematical and computational algorithms). This leads to a high submission rate of protein sequences to databases such as UniProt, which now contain millions of sequences [18] that can be used

[*] This work was partially supported by CNPq, CAPES, and FAPERJ.

in such analyses. Due to the need of processing such volume of data, the execution of this type of experiment became data- and CPU-intensive, thus requiring High-Performance Computing (HPC) environments to process data and analyze results in a timely manner.

A phylogenomic analysis experiment may be modeled as a workflow [4]. A workflow is an abstraction that allows for the user (*e.g.*, a bioinformatician) to compose a series of activities connected by data dependencies, thus creating a dataflow. These activities are typically legacy programs (*e.g.*, MAFFT, BLAST, *etc*). Bioinformaticians often fall back on Workflow Management Systems (WfMSs), such as Galaxy [2], Pegasus [5] and SciCumulus [12], to model and manage the execution of these workflows, but they have to rewrite the workflow into their languages and restrictions to their execution environments. Thus, despite collecting provenance data [6] (the derivation history of a data product, starting from its original sources - *e.g.*, a dataset containing many DNA and RNA sequences), which is a key issue to analyze and reproduce results (and allied with domain-specific data and telemetry data, it provides an important framework for data analytics), WfMSs are often not used by bioinformaticians.

This way, several bioinformaticians prefer to implement their experiments using scripts (*e.g.*, Shell, Perl, Python) [9]. More recently, bioinformaticians also started to explore efficient Data-Intensive Scalable Computing (DISC) systems and migrate their data- and CPU-intensive experiments to frameworks like Apache Spark (`https://spark.apache.org/`), *e.g.*, SparkBWA [1] and ADAM (`https://adam.readthedocs.io/en/latest/`).

Collecting provenance data (and domain-specific data) from scripts and DISC frameworks is challenging. There are several alternatives to WfMSs focused on capturing and analyzing provenance from scripts and DISC frameworks that can be applied in phylogenomic analyses [7,15]. However, they also present some limitations. The first one is that some approaches require specific languages (*e.g.*, noWorkflow [15] works only with Python) and specific versions of the framework (*e.g.*, SAMbA-RaP [7] requires a specific version of Apache Spark). Flexibility to define the level of granularity is also an issue. In general, automatic provenance data capturing generates fine-grained provenance, which commonly overwhelms bioinformaticians with a large volume of undesired data to analyze (*e.g.*, access to files and databases). On the other hand, automatic capturing coarse-grained provenance may not provide enough data for analysis (even existing WfMS provide non-flexible level of granularity). In addition, capturing domain-specific and telemetry data is also an issue in these approaches. We consider that If this integrated database (provenance, domain-specific data and telemetry data) is available, bioinformaticians can focus on analyzing just relevant data, reproduce the results and also observe a specific pattern to infer that something is not going well in the script at runtime, deciding to stop it or change parameters.

To address these issues, DfAnalyzer [17] was recently proposed to provide an agnostic way for scientists to define the granularity of the provenance data capture. The DfAnalyzer provenance database can be queried at runtime (*i.e.*, during the execution of the experiment), using a W3C PROV compliant data model. This portable and easy-to-query provenance database is also a step towards the reproducibility of bioinformatics experiments, regardless how they are implemented, since DfAnalyzer can be coupled to scripts, Spark applications and existing WfMSs. In this paper, we specialize DfAnalyzer to the context of phylogenomic experiments and present the benefits for bioinformatics data analyses and debugging. Thus, the main contribution of this paper is an extension of DfAnalyzer to collect telemetry (performance) data and its customization for the bioinformatics domain.

The remainder of the paper is organized as follows. Section 2 provides background concepts, describing the DfAnalyzer tool and discusses related work. Section 3 introduces the extensions and customizations in DfAnalyzer tool and presents the evaluations in a case study with a phylogenomic experiment. Finally, Section 4 concludes the paper and points future directions.

2 Background

2.1 DfAnalyzer: Runtime Dataflow Analysis of Scientific Applications using Provenance

DfAnalyzer [17] is a W3C provenance compliant system that captures and stores provenance during the execution of experiments, regardless of how they are implemented. DfAnalyzer is based on a formal dataflow representation to register the flow of datasets and data elements. It allows for analyzing and debugging dataflows at runtime. One important characteristic of DfAnalyzer is that it captures only relevant data (as defined by the user), thus avoiding overloading users with a large volume of low level data. DfAnalyzer captures “traditional” provenance data (*e.g.*, data derivation path) but also domain-specific data through raw data extraction, *e.g.*, a DNA sequence or the e-value. These characteristics are essential since experiments may generate massive datasets, while only a small sub-set of provenance and domain data is relevant for analysis [14]. The original architecture of DfAnalyzer has five components: (i) Provenance Data Extractor (PDE); (ii) Raw Data Extractor (RDE); (iii) Dataflow Viewer (DfViewer); (iv) Query Interface (QI); and (v) Provenance Database. The first two components are invoked by plugging calls on the script, while the other three have independent interfaces for the user to submit data analyses at runtime.

After deploying the DfAnalyzer, users are required to identify relevant data in their own script, model these data and instrument the code (add DfAnalyzer calls) in order that DfAnalyzer automatically captures data and populates the database. It is worth noticing that the database tables are automatically created based on the instrumentation performed in the code. This data identification is based on the following: Dataflow: a tag to identify the the dataflow that is being captured; Transformations: The data transformations that are part of the dataflow; Tasks: The smaller unit of processing, a transformation may be executed by several tasks; Datasets: Group of data elements consumed by tasks and transformations. A transformation consumes an output produced by other transformation. Data elements: The attributes that compose datasets. They represent either domain-specific data or input parameters of the experiment. Such information is inserted at the beginning of the script or program. Before inserting the tags on the script, it is needed to map these concepts to the script's dataflow. Identifiying the dataflow on the script is essential to represent the data transformations, dependencies, and data elements that need to be stored in the provenance database.

Listing 1 shows an example of code instrumentation in DfAnalyzer. To instrument the code, we have a 3-step process: (i) import DfAnalyzer packages in the script; (ii) define the prospective provenance (*i.e.*, the definitions of the dataflow and transformations — lines 2 to 17); and (iii) define the retrospective provenance (*i.e.*, activities and data elements to capture — lines 20 to 27). When the scripts start running each call sends to DfAnalyzer the prospective and retrospective data to be stored in the database. Fig. 1 presents a fragment of the provenance database schema. Each dataset identified in the script has an associated table in the database. It is worth noticing that this instrumentation is performed only once and the script may be executed several times after that. There are datasets, data elements, and data transformations that are typically used in phylogenomic workflows. To avoid this repetitive step for bioinformaticians and allow for a consistent data representation, this work provides specialized services for DfAnalyzer users in this domain.

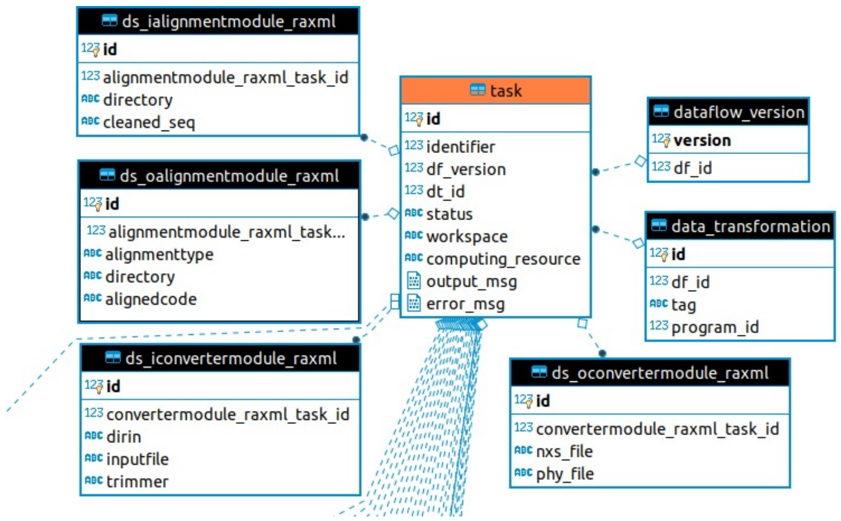


Fig. 1: A fragment of the provenance database schema

Listing 1 DfAnalyzer Prospective and Retrospective example code.

```
#define prospective model
dataflow_tag = "experiment_name"
df = Dataflow(dataflow_tag)
tf3 = Transformation("Alignment_"+dataflow_tag)
tf3_input = Set("iAlignment_"+dataflow_tag, SetType.INPUT,
    [Attribute("path", AttributeType.TEXT),
    Attribute("alignment", AttributeType.TEXT),
    Attribute("sequence", AttributeType.TEXT)])
tf3_output = Set("oAlignmentModule_"+dataflow_tag, SetType.OUTPUT,
        [Attribute("alignment", AttributeType.TEXT),
        Attribute("path", AttributeType.TEXT),
    Attribute("sequence", AttributeType.TEXT)])
tf3.set_sets([tf3_input, tf3_output])
tf2_output.set_type(SetType.INPUT)
tf2_output.dependency=tf2._tag
tf2.set_sets([tf2_output, tf3_input, tf3_output])
df.add_transformation(tf3)

#capturing retrospective data
t3 = Task(3, dataflow_tag, "Alignment_"+dataflow_tag, "t3")
t3_input = DataSet("iAlignment_"+dataflow_tag, [Element([dirin, alignment, corrected_file])])
t3.add_dataset(t3_input)
t3.begin()
        oalignment = execute_alignment(dirin, corrected_file, alignment, trimmer)
t3_output = DataSet("oAlignment_"+dataflow_tag, [Element([alignment, dirin, oalignment])])
t3.add_dataset(t3_output)
t3.end()
```

2.2 Related Work

Hondo *et al.* [8] use provenance to the analysis of the transcriptome of the fungus Schizosaccharomyces pombe in four different conditions. Like in DfAnalyzer, they also adopt the PROV-DM model [11], but use different noSQL database systems to represent provenance data. Despite the flexibility of PROV-DM, the data granularity level chosen in [8] is coarse-grain, just capturing the transformation name, the program used to execute it, and the transformation execution time. Afgan *et al.* [2] focus on providing support in RNA sequencing experiments in Galaxy WfMS for non-specialists. They encapsulate the complexity of the environment configuration and data analysis. Although the tool supports the design and execution of workflows, provenance capture has a fixed granularity level and domain-specific data is not captured.

Carvalho *et al.* [3] propose an approach that converts code from interactive notebooks into workflows to capture provenance and identify the dataflow. The approach identifies transformations and tasks automatically, *e.g.*, functions are transformations. Although this reduces the instrumentation effort, it is dependent on the organization of the code, *i.e.*, if the programmer does not use functions, the identification of the transformations may be compromised. In addition, it does not capture domain-specific data neither supports parallel scripts (which is very common in the bioinformatics domain). Pimentel *et al.* [15] propose the noWorkflow tool to automatically collect provenance from the execution of Python scripts. noWorkflow is easy to deploy, but it is specific for Python scripts and does not support parallel executions. Pina *et al.* [16] also specialized DfAnalyzer, but they have focused on fine-tuning parameters in scripts of Convolutional Neural Networks. It does not help capturing provenance in the genomics domain neither telemetry data.

3 Evaluating DfAnalyzer for Phylogenomics Experiments

This section presents the extensions implemented in DfAnalyzer to be specialized for the phylogenomics domain and the evaluation with the SciPhylomics workflow [13].

3.1 Specializing DfAnalyzer

DfAnalyzer already provides ways to capture provenance and domain-specific data, but based on previous experiments, performance data are also very important for data analytics. Bioinformaticians often have to analyze the domain-specific data together with the performance data to evaluate the trade-off between producing results with high quality and the time needed to produce such results.

Therefore, in this subsection we explain the specialization of DfAnalyzer to capture performance data. We have added a new component to the DfAnalyzer architecture named *Telemetry Data Extractor*. This component is build on top of the *psutil* library (version 5.7.2). Psutil is cross-platform and leads to retrieve performance data of running processes. It also leads to capture system resources usage (*e.g.*, CPU, memory, *etc.*). Before the execution of each task or transformation, the TDE component is invoked to monitor the resource usage. The monitoring process ends when the task or transformation finishes execution. All performance data are stored in a specific table that is associated to the *Task* table presented in Fig. 1.

3.2 SciPhylomics Workflow

SciPhylomics [13] is a phylogenomic analysis workflow that aims to construct phylogenetic trees comparing hundreds of different genomes. SciPhylomics is composed by nine transformations. The first four transformations are associated to phylogenetic analysis (or gene phylogeny): (i) multiple sequence alignment (MSA), (ii) MSA conversion, (iii) search for the best evolutionary model, and (iv) construction of phylogenetic trees. After the execution of the gene phylogeny activities, the data quality analysis activity is executed. This data quality analysis allows for filtering results that do not meet a quality criteria. The last four activities represent the phylogenomic analysis (or genome phylogeny): (vi) concatenation of MSA to obtain a "superalignment", (vii) election of an evolutionary model for the "phylogenomic tree" construction based on the "superalignment", (viii) construction of phylogenomic trees and (ix) the phylogenomic tree election.

SciPhylomics is data- and CPU-intensive, so it requires HPC environments to produce results in a feasible time. In this evaluation, we implemented SciPhylomics in a Python script that is build on top of Parallel Python library. Parallel

Python is a python module which provides mechanisms for parallel execution of python scripts on multiple processors/cores and clusters.

As aforementioned, the definition of the dataflow is essential for identifying capturing relevant provenance, domain-specific and telemetry data. This way, in this evaluation we considered a fragment of Sciphylomics composed of five transformations associated to gene phylogeny, as presented in Fig. 2: (i) Sequence Cleaning; (ii) Sequence Alignment; (iii) Sequence Conversion (iv) Model Generation; (v) Tree Generation. It is worth noticing that the transformations of the dataflow were defined specifically for the context of this paper, being possible to split these transformations into smaller ones (or merging them into a bigger one).

After defining the transformations in the script, we need to list the programs that execute them. In this fragment of SciPhylomics, each transformation may be implemented by several programs (*i.e.*, a variability). The gray transformations presented in Fig. 2 are the variant transformations, *e.g.*, data transformation (2) can be executed by the applications MAFFT, ClustalW, and Muscle. Similarly, in transformation (5) the programs RAxML and MrBayes can be used.

This characteristic is interesting for the analysis phase since we can compare the results and performance of different programs that implement the same transformation. After identifying SciPhylomics structure and the variabilities, we instrumented the SciPhylomics python script according to Listing 1 to specify the data transformations, tasks, data dependencies, datasets, and data elements.

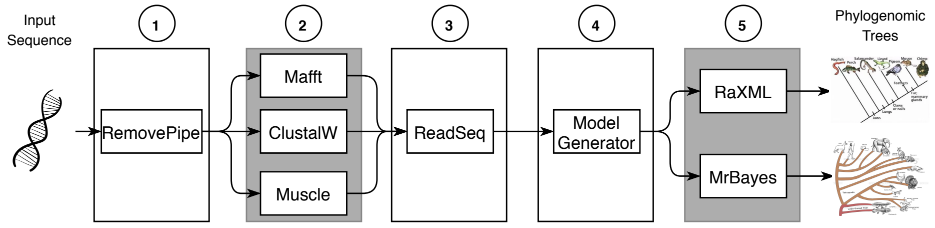


Fig. 2: A fragment of the SciPhylomics dataflow

3.3 Setup of the Experiment

The fragment of SciPhylomics presented in Fig. 2 was executed varying the programs in transformations 2 and 5. We defined two different executions of SciPhylomics (called Experiments A and B) depending on the chosen tree generation program. In addition, for each experiment, we can vary the chosen MSA program. Thus, six variants were performed with a dataset composed of 98 different multifasta files (each variant was executed 10 times).

The machine configuration used to execute SciPhylomics was an AMD FX(tm)-8150 8-Core Processor, 32GB RAM, and 2TB hard disk. The overview of the parameter values used in the executions is in Table 1. The parameters values were defined following an specialist. In special, *Ngen* is related to the number of cycles that the Markov Chain Monte Carlo (MCMC) algorithm is executed, the main algorithm goal is to make small random changes in some parameters to accept or reject according to the probability. Also, *Nchains* is the parameter related to the number of different parallel MCMCMC chains within a single execution run, *printfreq* is related to the frequency that the information is shown on the screen, *nruns* how many independent analyses are started simultaneously. Concerning the MSA programs, it was not necessary to set parameters and the programs were executed in default mode. More information are find on Muscle (`www.ebi.ac.uk/Tools/msa/muscle/`), Mafft (`mafft.cbrc.jp/alignment/software/`), Clustalw (`www.genome.jp/tools-bin/clustalw`), RAxML (http://cme.h-its.org/exelixis/web/software/raxml/) and MrBayes (`http://mrbayes.sourceforge.net/commref_mb3.2.pdf`) documentation.

Table 1: Parameters of the variants of SciPhylomics fragment

Experiment A			**Experiment B**			
Wf. Step	**Program**	**Parameter**	**Wf. Step**	**Program**	**Parameter**	
Clean	RemovePipe	Total	Clean	RemovePipe	Total	
Alignment	Mafft	Default	Alignment	Mafft	Default	
	ClustalW	Default		ClustalW	Default	
	Muscle	Default		Muscle	Default	
Converter	ReadSeq	Default	Converter	ReadSeq	Default	
Evolutive Model Generator	Model Generator	Default	Evolutive Model Generator	Model Generator	Default	
Tree Generator	RaXML	Default	Tree Generator	MrBayes	ngen	100000
					nchains	4
					printfreq	1000
					burnin	0
					nruns	2
					rates_mrbayes	4

3.4 Data Analysis in Practice

In this Section, we aim at presenting the advantages of analyzing provenance from phylogenomic analyses in practice using a series of analytical queries. The experimental process was executed in 75.08h. For the analysis presented in this section, we used six DfAnalyzer tables: *dataflow* (contains information about the dataflow), *data_ set* (contains details of the datasets), *data_ transformation* (contains details of the transformations), *task* (contains details of the tasks), *dataflow_ execution* (contains statistics of the execution of the dataflow), *data_ transformation_ execution* (contains statistics of the execution of each transformation). For more details about the DfAnalyzer database schema please refer to [17].

The first analysis is related the evolutionary model generated by the fourth transformation, implemented by ModelGenerator program. Let us assume that the user needs to analyze what are the models chosen for the input data. If the bioinformatician executed SciPhylomics without DfAnalyzer, one has to open each file to check this information, which is time-consuming, tedious, and error-prone. By using DfAnalyzer, one has only to submit the query shown on Listing 2. In this query, the user wants to discover the number of times a specific evolutionary model was used, but just when the length of the input sequence is larger than 20.

Listing 2 Number of occurrences of a specific evolutionary model

```
SELECT count(*) as occurence, model
                FROM ds_omodelgeneratormodule_mrb
                GROUP BY model
                ORDER BY occurence DESC;
```

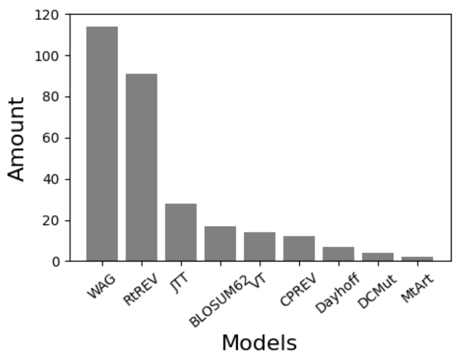
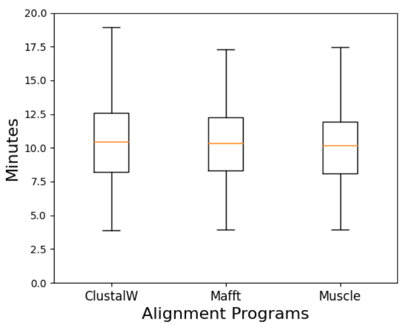
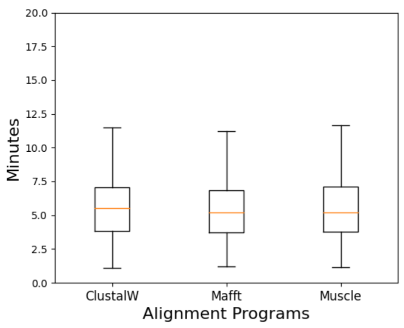


Fig. 3: Number of occurrences of each evolutionary model.

The result of the query presented in Listing 2 is shown in Fig. 3. One can state that WAG and RtREV models are the most common ones. WAG presented higher likelihoods than any of the other models. This type of query can be adapted to other attributes of the database, such as quality of the generated tree, e-value, etc. Another performed analysis was related to the execution time of each transformation and resource usage. Capturing this type of data can impact the performance of the experiment since it is usually captured in a short time interval and may introduce overhead. Thus, in this analysis we defined an interval for capturing performance data (30 second window). The box-plots of the execution time behaviour for the six variations of SciPhylomics are presented in Fig. 4.



(a) SciPhylomics with MrBayes



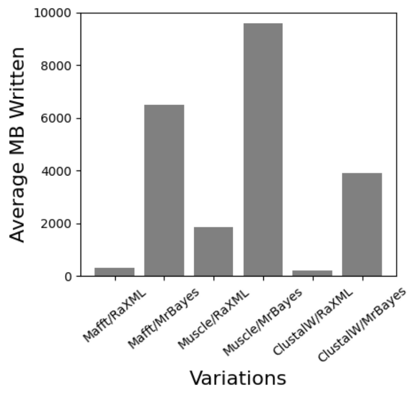
(b) SciPhylomics with RaXML

Fig. 4: Execution time of SciPhylomics varying the MSA programs (ClustalW, Mafft and Muscle) and tree generator programs (MrBayes and RaXML).

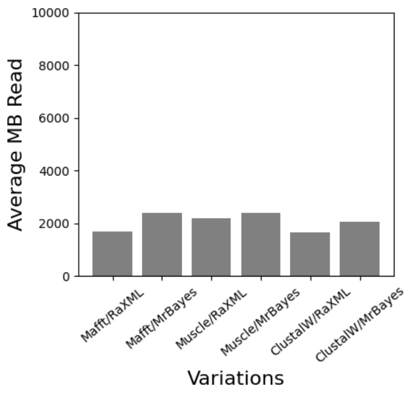
As presented in Fig. 4, there is a non-negligible variation in execution time according to the tree generator program used in the experiments. The experiments executed with RaXML (Fig. 4b) finished faster than those performed with MrBayes (Fig. 4a). The execution time difference can be explained based on the technique used for each program. MrBayes performs a Bayesian inference and model choice across a wide range of phylogenetic and evolutionary models and is more costly than RAxML. The programs execute different operations and generate different datasets that are composed of different attributes as well. While RaXML does not execute the operation of "search for evolutionary model"(since the ModelGenerator program is executed), MrBayes execute this operation, and in this experiment variability the operation "search for evolutionary model" is executed twice. These characteristics explain the difference in time execution in both cases, and the resource usage as well, as shown on Fig. 5.

4 Conclusion

This paper presents an approach to capture provenance data from phylogenomic dataflows without the need to rewrite the workflow to a specific programing language or Workflow Management System engine. We specialized the novel DfAnalyzer tool in the context of Phylogenomic experiments so that in an existing workflow it is possible to achieve a flexible granularity-level data capture and create an integrated database composed of domain-specific, provenance and telemetry data. DfAnalyzer is prospective provenance based. By modelling prospective



(a) Memory Usage



(b) Disk Usage

Fig. 5: SciPhylomics Resource usage: Memory and Disk

provenance data, retrospective provenance is automatically captured while the workflow executes, and the data created by its transformations are stored in a relational database for further querying. Differently from other approaches, the provenance is captured with flexible granularity, and the bioinformaticians can specify what is important for their analysis and reduce the experiment cost in different spheres.

Another advantage of applying DfAnalyzer in the context of Phylogenomic experiments is the capability of capturing data from experiments that execute either locally or HPC environments, due to the fact that DfAnalyzer is asynchronous and request-based, it can execute in different environments. This asynchronous characteristic contributes that the instrumentation does not cause delays in the workflow execution. In addition, we extended DfAnalyzer to capture telemetry (performance data). This way, users are allowed to perform analyses based on both the provenance data and performance metrics. We evaluated the proposed specialization of DfAnalyzer using the previously defined SciPhylomics workflow and the proposed approach showed relevant telemetry scenarios and rich data analyses. In future work, we intend to evaluate reproducibility in experiments based on the analysis of the provenance database.

References

1. Abuín, J.M., Pichel, J.C., Pena, T.F., Amigo, J.: SparkBWA: Speeding up the alignment of high-throughput DNA sequencing data. PLOSONE **11**(5), e0155461 (2016)
2. Afgan, E., Baker, D., Van den Beek, M., Blankenberg, D., Bouvier, D., Čech, M., Chilton, J., Clements, D., Coraor, N., Eberhard, C., et al.: The galaxy platform

for accessible, reproducible and collaborative biomedical analyses: 2016 update. Nucleic acids research **44**(W1), W3–W10 (2016)

3. Carvalho, L.A.M.C., Wang, R., Gil, Y., Garijo, D.: Niw: Converting notebooks into workflows to capture dataflow and provenance. In: Tiddi, I., Rizzo, G., Corcho, Ó. (eds.) Proceedings of Workshops and Tutorials of the 9th International Conference on Knowledge Capture (K-CAP2017), Austin, Texas, USA, December 4th, 2017. CEUR Workshop Proceedings, vol. 2065, pp. 12–16. CEUR-WS.org (2017)
4. de Oliveira, D.C.M., Liu, J., Pacitti, E.: Data-Intensive Workflow Management: For Clouds and Data-Intensive and Scalable Computing Environments (2019)
5. Deelman, E., et al.: Pegasus, a workflow management system for science automation. FGCS **46**, 17–35 (2015)
6. Freire, J., Koop, D., Santos, E., Silva, C.T.: Provenance for Computational Tasks: A Survey. CS&E **10**(3), 11–21 (May 2008)
7. Guedes, T., Martins, L.B., Falci, M.L.F., Silva, V., Ocaña, K.A.C.S., Mattoso, M., Bedo, M.V.N., de Oliveira, D.: Capturing and analyzing provenance from spark-based scientific workflows with samba-rap. Future Gener. Comput. Syst. **112**, 658–669 (2020)
8. Hondo, F., Wercelens, P., da Silva, W., Castro, K., Santana, I., Walter, M.E., Araújo, A., Holanda, M., Lifschitz, S.: Data provenance management for bioinformatics workflows using nosql database systems in a cloud computing environment. In: 2017 IEEE International Conference on Bioinformatics and Biomedicine (BIBM). pp. 1929–1934. IEEE (2017)
9. Marozzo, F., Talia, D., Trunfio, P.: Scalable script-based data analysis workflows on clouds. In: WORKS. pp. 124–133 (2013)
10. Masulli, F.: Computer Methods and Programs in Biomedicine **91**(2), 182 (2008)
11. Moreau, L., Missier, P., Belhajjame, K., B'Far, R., Cheney, J., Coppens, S., Cresswell, S., Gil, Y., Groth, P., Klyne, G., et al.: PROV-DM: The PROV Data Model. W3C Recommendation **30**, 1–38 (2013)
12. Oliveira, D., Ocaña, K.A.C.S., Baião, F.A., Mattoso, M.: A provenance-based adaptive scheduling heuristic for parallel scientific workflows in clouds. JGC **10**(3), 521–552 (2012)
13. de Oliveira, D., Ocaña, K.A.C.S., Ogasawara, E.S., Dias, J., de A. R. Gonçalves, J.C., Baião, F.A., Mattoso, M.: Performance evaluation of parallel strategies in public clouds: A study with phylogenomic workflows. Future Gener. Comput. Syst. **29**(7), 1816–1825 (2013)
14. Olma, M., Karpathiotakis, M., Alagiannis, I., Athanassoulis, M., Ailamaki, A.: Slalom: Coasting through raw data via adaptive partitioning and indexing. Proceedings of the VLDB Endowment **10**(10), 1106–1117 (2017)
15. Pimentel, J.F., Murta, L., Braganholo, V., Freire, J.: noworkflow: a tool for collecting, analyzing, and managing provenance from python scripts. Proc. VLDB Endow. **10**(12), 1841–1844 (2017)
16. Pina, D.B., Neves, L., Paes, A., de Oliveira, D., Mattoso, M.: Análise de hiperparâmetros em aplicações de aprendizado profundo por meio de dados de proveniência. In: Anais Principais do XXXIV Simpósio Brasileiro de Banco de Dados. pp. 223–228. SBC (2019)
17. Silva, V., de Oliveira, D., Valduriez, P., Mattoso, M.: Dfanalyzer: runtime dataflow analysis of scientific applications using provenance. Proceedings of the VLDB Endowment **11**(12), 2082–2085 (2018)
18. The UniProt Consortium: UniProt: the universal protein knowledgebase. Nucleic Acids Research **45**(D1), D158–D169 (11 2016)