paper:7

Debugging Scientific Workflows with Provenance: Achievements and Lessons Learned

Daniel de Oliveira[1], Flavio Costa[2], Vítor Silva[2], Kary Ocaña[2] and Marta Mattoso[2]

[1]Institute of Computing – Fluminense Federal University (IC/UFF)

[2]COPPE – Federal University of Rio de Janeiro (UFRJ)

danielcmo@ic.uff.br, {flscosta, silva, kary, marta}@cos.ufrj.br

Abstract. *Scientific Workflow Management Systems manage experiments in large-scale and deliver provenance data. Provenance data represents the workflow execution behavior, allowing for tracing the data-flow generation. When provenance is extended with performance execution data, it becomes an important asset to identify and analyze errors that occurred during the workflow execution (i.e. debugging). Debugging is essential for workflows that execute in parallel in large-scale distributed environments since the incidence of errors in this type of execution is high and difficult to track. By debugging at runtime, scientists can identify errors and take the necessary actions, while the workflow is still running. We present provenance based debugging, in real use cases, running in parallel, with virtual machines in clouds. In these experiences scientists use provenance data to query domain and execution data to detect errors, especially when the execution concludes without any system error message.*

1. Introduction

Many scientific experiments are modeled as scientific workflows to represent the flow of data, transformed by programs, each of which consumes a set of parameters and input data. Scientific workflows are managed by powerful engines in Scientific Workflow Management Systems (SWfMS), such as Pegasus (Lee *et al*. 2008), Swift/Turbine (Wozniak *et al*. 2012), Tavaxy (Abouelhoda *et al*. 2012) and SciCumulus (Oliveira *et al*. 2010), which have parallel execution support. SWfMS are used in domains, such as, engineering, bioinformatics and astronomy (Hey *et al*. 2009). Furthermore, several scientific workflows execute in large scale, since they process large amounts of data (Pennisi 2011) requiring parallel execution in High Performance Computing (HPC) environments, such as clusters, grids, and more recently, clouds. These SWfMS also provide provenance capabilities to record the historical information about the workflow execution behavior (Freire *et al*. 2008) to help workflow analysis, reproducibility and validation. Although several SWfMS provide sophisticated mechanisms for executing large-scale scientific workflows in distributed HPC environments, most of them execute the workflow in an "offline" way (Ailamaki *et al*. 2010), i.e. as a black-box. Online monitoring and debugging may save significant amounts of workflow execution time, when unexpected behavior can be detected way before the end of workflow execution. When debugging can analyze the relation between execution data, workflow data and

[1] This work is partially sponsored by CNPq and FAPERJ.

domain data, scientists have a much clear picture of the experiment dataflow (Mattoso *et al*. 2013). When this analysis is done at runtime, significant time may be saved.

We consider here three types of unexpected execution behavior, one is related to execution performance, the second is aware of the workflow stage of execution and the third is related to data-flow generation, including domain data analysis (Gonçalves *et al*. 2012). Execution performance may indicate that a task is taking longer than expected. It also identifies failures from node, task or program crashes, among others. Workflow execution may be unexpected when a data is generated and does not produce the expected result value. In fact, this may prevent the next activity to consume it, generating an anomaly. For example, the specification of a filter value may not be adequate, generating an excessive or too restrictive data-flow. Since these results would not cause any failure on the execution, scientists would only realize the filter configuration problem by the end of the whole workflow execution. By further analyzing domain data, scientists may evaluate if domain data is not compatible with the defined filter value. Therefore, data-flow monitoring also helps to improve workflow performance and should be together with domain data and execution monitoring.

To help workflow execution monitoring and debugging, related work like Stampede (Gunter *et al*. 2011) and Panda (Ikeda *et al*. 2012) support data-oriented scientific workflow analysis using runtime execution data. These systems are focused on workflow activities and their corresponding jobs and tasks executions, using performance analyzers, such as, NetLogger to associate execution behavior to activities. They represent performance metrics and file generations in a relational database so that users can query and identify execution failures in scientific workflows, at runtime, even when the workflow execution does not crash. Stampede has a visual interface where execution behavior may be visualized and analyzed with R package (Vahi *et al*. 2013).

Nevertheless, these powerful execution performance monitoring and debugging do not allow for domain data-flow generation debugging. This would require access to provenance data-flow and some related domain data, also at runtime. When provenance data and execution monitoring data are in separate databases, it is difficult to relate execution behavior with the data that caused an unexpected activity execution. For example, if the user detects that a specific task failed or is taking too long, it might be helpful to check the inputs and parameters of this task, or in which workflow iteration this failure occurred. To track this, the user would need first to query the execution monitoring database to find the instance of the failed task, and then query the domain data-flow, to manually try to relate them all together, *i.e*., the execution instance task with its corresponding input data-flow.

In another related work, MTCProv (Gadelha *et al*. 2012) stores performance execution time in the same database as provenance data and it helps scientists in data-flow debugging through provenance queries. However, in MTCProv, online monitoring is not supported, therefore debugging queries can only be submitted after the workflow finishes its execution. Unexpected data-flow might not generate a failure occurrence, however it would still represent an anomalous execution requiring aborting the execution as soon as possible. Waiting until the end to debug, considering performance and data-flow, change the workflow specification and resubmitting it takes too long.

Differently from the current mainstream, Chiron (Ogasawara *et al*. 2011) and its extension to cloud environments, SciCumulus, are the only SWfMS that support

debugging based on execution performance and domain data-flow generation. They obtain that by integrating all these data at their provenance database, available for traversal analysis at runtime. In this paper we go deep into specific details of the data-flow algebraic approach (Ogasawara *et al*. 2011), adopted in both Chiron and SciCumulus, to show the potential of data analysis in real applications. Therefore, this paper presents our achievements and lessons learned from users exploring the runtime provenance support of SciCumulus in debugging real large-scale bioinformatics workflows (Ocaña *et al*. 2011, Ocaña *et al*. 2014).

This paper is organized as follows: Section 2 explains the importance of provenance in scientific workflow monitoring and debugging; Section 3 describes case of studies for workflow debugging at runtime using SciCumulus; and finally, Section 4 presents our conclusions and future research directions.

2. Data-flow Runtime Analysis

The workflow data centric algebraic approach (Ogasawara *et al*. 2011), that drives the execution engine of both Chiron and SciCumulus, uses relations to store the workflow specification as a prospective provenance and the workflow execution as a retrospective provenance, all in the same database. In other words, prospective provenance presents a workflow composition (such as activities and your dependencies), while retrospective provenance determines properties about workflow execution (such as the directory to execute workflow or database hostname) (Freire *et al*. 2008). This algebraic approach introduces a set of operators to the relational algebra (*Map*, *Reduce*, *Filter*, *SplitMap*, *MRQuery* and *SRQuery*), which considers both workflow activities and data-flow as operands. Each activity of the workflow is associated to an operator, which has operand relations as input and all results of this activity execution are stored at an output relation, which can be the input operand relation to the operator associated to the next workflow activity.

To present the potential of monitoring and debugging, we show the algebraic view of two real workflows from the bioinformatics domain using SciCumulus. One is SciPhy, for phylogeny analyses (Ocaña *et al*. 2011) and the other is SciDock for molecular docking analyses (Ocaña *et al*. 2014). SciPhy and SciDock are complex workflows composed by several data and computing-intensive activities that may last for days, weeks or even months, to generate results. We simplify them here and show the algebraic relations only for the most representative activities (programs) for both workflows: RAxML for SciPhy and autodock for SciDock.

SciPhy, partially represented in Figure 1, is modeled as two algebraic expressions: $R_m \leftarrow$ *Map*(RAxML, R_i); $R_o \leftarrow$ *Reduce*(TreeView, "model", R_m). RAxML is followed by the phylogenetic analysis activity TreeView. Workflow parameters from the *Command line* are now attributes of input relation R_i and their values are inserted in R_i tuples (Figure 1.a). RAxML is executed n times with different input data for three parameters (*i.e*. the last attributes in R_i) generating n resulting tuples in the output relation R_m. Each tuple of R_m is related to the consumption of each R_i tuple (Figure 1.b). Attribute k relates R_i with R_m. TreeView executes a computational simulation by grouping R_m tuples by *model* (*Reduce*) into output relation R_o to obtain a phylogenetic tree and for analyzing/visualizing the quality of the tree (Figure 1.c).

Scientists may query R_m at runtime, while the output tuples are still being generated. For example, a query may retrieve tuples with attributes *Result = true* and *model = GTR*, which were generated from a runtime calculated average bootstrap attribute value in R_i that are above a threshold. This would indicate an issue in the obtained model that will influence the topology of the trees. In another example, based on data-flow analysis, scientists decide that they are only interested on result models that generate trees where bootstrap *quality-bp (AVG) > 90*, and they may submit relational queries to R_o to filter out unwanted results for the next steps. Estimating adequate thresholds or filters before evaluating the data-flow is complex and anticipating the "right" value might not be possible.

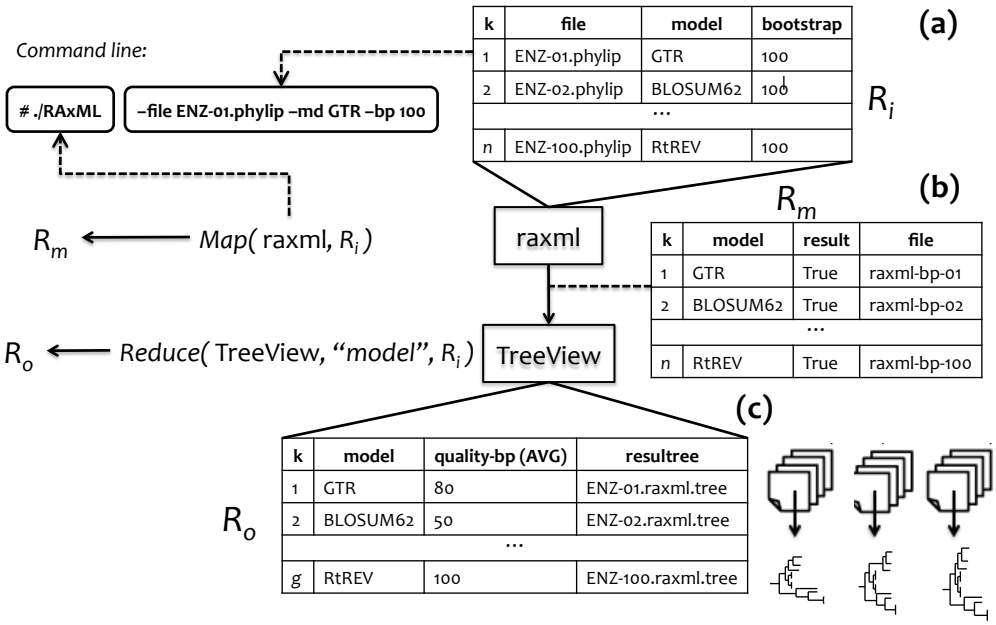

Figure 1. Reduced algebraic workflow representation of SciPhy

SciDock is a workflow for molecular docking that attempts to mimic the process of bringing together a protein and a ligand to form a non-covalent complex and to reveal the electrostatic and steric complementarity between the protein and the ligand. Figure 2 shows the most representative activities of SciDock (autodock4) and the following docking analysis (Vina). Vina executes a computational simulation for analyzing/ visualizing the quality of the molecular docking for the receptor-ligand pairs (True, if it exists/False, otherwise), reinforcing the biological inferences attributed to each obtained molecular docking and statistical results. The algebraic workflow would be: $R_x \leftarrow Map$ (autodock4, R_e); $R_s \leftarrow Map$ (Vina, R_x) ; represented by Figure 2.a and Figure 2.b.

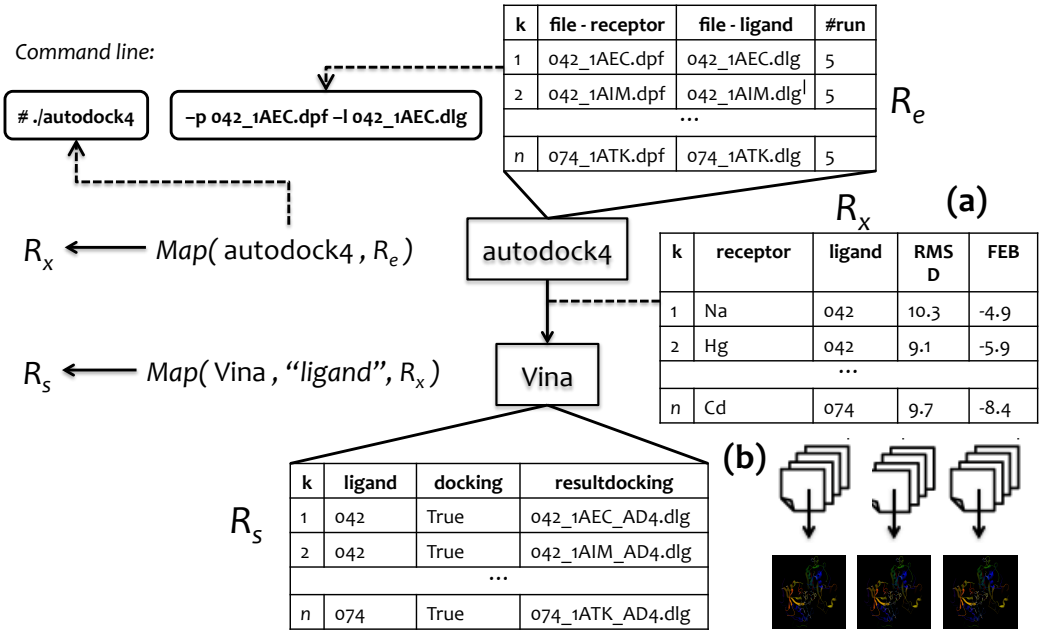

Figure 2. Reduced algebraic workflow representation of SciDock

Scientists may query R_x data-flow at runtime and check if it presents negative average FEB values and RMSD > 4 for any tuple. This would indicate an issue with the obtained ligand/receptor biophysical characteristics that will influence the existence (or not) of the docking process, depending on some the quality values (*i.e.*, FEB, RMSD).

3. Debugging Bioinformatics Workflows Using Execution and Domain Data

During SciPhy and SciDock executions, SciCumulus provenance database was queried to detect possible unexpected situations from workflow executions. We discuss here queries related to performance execution time and data-flow generation for detecting domain-specific problems that would not be identified using monitoring and debugging mechanisms based only on performance execution data.

Domain-data relations with parameters consumed and produced by activities (*i.e.* R_i and R_m) are stored in the same provenance database that has execution data and typical workflow data. The workflow algebraic approach fostered this provenance data integration. Our provenance relations follow the PROV-Wf model (Costa *et al.* 2013), an extension of the W3C PROV standard (Groth and Moreau 2013). Using PROV-Wf, scientists are able to query performance execution time, domain and data-flow from workflow execution, as soon as they are generated, at runtime. PROV-Wf is composed by a set of classes (each one has an associated relation at the provenance database) to represent workflow concepts. In Figure 3 we show some of them, used in our queries: *hworkflow*, *hactivity* for prospective provenance and *hactivation*, *relation*, *hfile*, and *hkeyspace* for retrospective provenance. The instance of one *hactivity* of an *hworkflow* execution is represented as a tuple in *hactivation* through attribute *actid*. Table *relation* has its name and attributes defined by the user to store domain data values for input parameters that are consumed by an activity execution, like R_i is input to RAxML, and *hfile* provides access from activities to domain files, like ENZ-01.phylip in R_i and raxml-bp-01 in R_m.

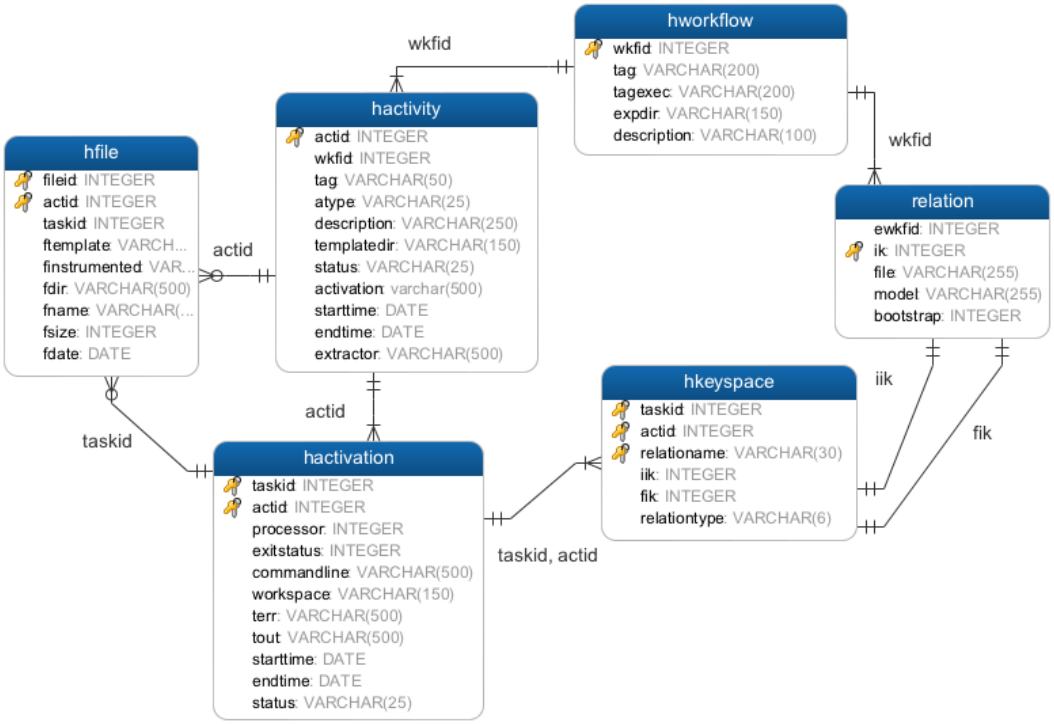

Figure 3. Simplified view of PROV-Wf classes, adapted from (Costa *et al.* 2013)

All experiments presented in this section were performed in the Amazon EC2 cloud environment using 32 Amazon's large VMs (EC2 ID: m1.large – 7.5 GB RAM, 850 GB storage, 2 cores). Provenance data is managed using PostgreSQL database system, stored at Amazon S3. The execution of SciPhy generated 1,000 activations for 200 input files. SciDock generated 1,798 activations for 200 input multi-fasta files. This means that for each execution of SciPhy 1 row is inserted in *hworkflow* table, 5 in *hactivity*, 1,000 in *hactivation*, 1,000 in *relation*, 1,000 in *hkeyspace* and 5,315 in *hfile*. In the case of SciDock, for each execution consuming 200 input data, 1 row is inserted in *hworkflow*, 9 in *hactivity*, 1,798 in *hactivity*, 1,798 in *hrelation*, 1,798 in *hkeyspace* and 10,380 rows in hfile. In total, the provenance database contains the following amount of rows *per* table: *hworkflow:* 757, *hactivity:* 3265, *hactivation:* 67,039, *hfile:* 272,218, *relation:* 67,035, and *hkeyspace:* 67,035. The following selected queries were considered in our analysis to represent three types of the most frequent queries: execution, data-flow and monitoring queries. In this case, our monitoring query aims to identify activity executions with errors.

Query 1 (Q1) – Execution time. SciPhy workflow is executed 1,000 times, varying a large number of parameters or data input. Errors can be identified at runtime, when scientists are monitoring and tracing one specific task. Using task execution time information, from historic provenance, users are able to identify performance variations that may indicate execution outliers. If one task is consuming more time to execute than expected (*e.g.* more than average execution time), there is an indication of alert. Scientists have to investigate if this performance variation is an error. Related data files are identified through provenance queries and then analyzed. In SciPhy, queries may check: the number of sequences in the input fasta file (*i.e.*, if there are less than three or more than 100 fasta sequences); if the format (for amino acid sequences) is recognized; or if the default number for bootstrap replication is 100. When scientists are able to

analyze performance metrics with domain-specific data, they can evaluate if the execution is likely to be correct or not. This execution time debugging with domain data saved 15% of time, since waiting until the end of the execution would require additional 3.1 hours (Ocaña *et al*. 2011).

```
Q1: SELECT w.tag, a.tag, t.taskid, t.exitstatus, t.processor, t.workspace, t.status, t.endtime, t.starttime,
    extract ('epoch' from (t.endtime-t.starttime))||',' as duration, r.file, r.bootstrap
    FROM hworkflow w, hactivity a, hactivation t, hkeyspace k, relation r
    WHERE w.wkfid = a.wkfid  AND a.actid = t.actid AND t.taskid = k.taskid AND r.ik >= k.iik
    AND r.ik <= k.fik
    AND w.tag like '%SciPhy%'
    AND r.bootstrap <> 100
    AND duration > (SELECT avg(extract ('epoch' from (t.endtime-t.starttime))||',')-
                        2.4*stddev(extract('epoch' from (t.endtime-t.starttime))||',')
                        FROM hactivation ac WHERE ac.actid=t.actid)
```

Query 2 (Q2) – Data-flow. In SciDock workflow, approximately 30% of executions failed. Problems involving environment and workflow specification were checked and discarded (*i.e*. any problems related to virtual machines or environment configuration). Therefore, we concluded that failures were associated with input data (*i.e*. ligands or receptor structures). Querying the provenance database, the errors were constantly traced back to a specific set of ligand structures (*i.e*. very small ligands), which were not compatible with receptors of the cysteine protease family. Then, these ligands were detected and excluded for *a posteriori* executions. At the end, results were re-verified with Q2 to confirm that the ".dlg" files (containing the final docking results) were correctly generated (*i.e*. as Successful Completion), and consequently, the docking quality scores (*e.g*. FEB and RMSD) analyzed by specialists. This data-flow query saved 30% of time, since waiting the execution end would require another 21.2 hours. However, even after waiting for the workflow completions, if this direct access to the set of ligand structures was not available, it would be very difficult to identify that the anomalous ligands all shared a common property, related to the failures.

```
Q2: SELECT w.tag, a.tag, f.fname, f.fsize, f.fdir, r.RMSD, r.FEB
    FROM hworkflow w, hactivity a, hactivation t, hfile f, hkeyspace k, relation r
    WHERE w.wkfid = a.wkfid AND a.actid = t.actid AND f.taskid = t.taskid AND t.taskid = k.taskid
    AND r.ik >= k.iik AND r.ik <= k.fik
    AND w.tag like '%SciDock%' AND f.fname like '%dlg%'  AND t.exitstatus = 0
```

Query 3 (Q3) Unexpected program errors. By default, programs like AutoDock and AutoDock Vina do not recognize a set of atoms (*e.g., Na, Hg, Ac, Cd, Ce, V, K*) producing the following error message: "*Unknown receptor type: 'Hg'*". The problem is that users cannot specify *a priori* which atoms are inside each ligand/receptor structure. However, in structures of enzyme families, as cysteine protease, used in our experiments, some receptor structures can contain mercury (*Hg*). This scenario was detected by querying the provenance database at runtime. Q3 searches for tasks that present some failures in the execution (*i.e*. that produced error messages as outputs) and whose input data has the atom *Hg* in its cysteine protease structures. When identifying these tasks, the *Hg* molecule was included to be recognized in cysteine protease receptors. Finally, all executions that failed due to this problem were successfully identified and re-executed. This data-flow query saved 3% of time, since waiting until the end of the execution would require 3.9 hours. Again, debugging without provenance would be even more time consuming.

```
Q3: SELECT w.wkfid, w.tag
    FROM hworkflow w, hactivity a, hactivation t, hkeyspace k, relation r
    WHERE w.wkfid = a.wkfid AND a.actid = t.actid AND t.taskid = k.taskid AND r.ik >= k.iik
    AND r.ik <= k.fik
    AND w.tag like '%SciDock%'
    AND a.tag = 'autogrid4'
    AND r.receptor == 'Hg'
    AND (t.terr <> '' OR t.exitstatus <> 0)
```

The unexpected behavior detected by queries 1, 2 and 3 could only be identified by associating performance execution data with provenance data and domain-specific data. Using relational queries, scientists can check on thousands of executions related to hundreds of thousands of domain data files. This represents a step forward to scientists regarding related work. While querying the integrated database at run time, users do not have to manually find the required data files, download them and analyze each file individually (without data flow context) to detect failures and debug unexpected behavior in their workflow execution. Typical queries are also available as simpler parameterized query templates.

4. Conclusions and Final Remarks

Debugging mechanisms in HPC scientific workflows are essential to support the exploratory nature of science and the dynamic process involved in scientific analyses. Large-scale experiments can benefit from debugging features to reduce the incidence of errors, decrease the total execution time and sometimes reduce the financial cost involved. In most of the executions, typical HPC profiling tools are too low level, not aware of workflow resources. The debugging process has to explore the content of input data, iteratively searching for the identification of what caused the anomalous execution. Since each workflow execution may consist of hundreds or even thousands of activities that are executed in parallel, it is unviable to perform a manual monitoring and debugging on data-flow generation.

This paper discussed the importance of W3C provenance data enriched with performance and domain-specific data. Furthermore, real bioinformatics workflow executions present how to make fine-tuning debugging using the provenance data at runtime. The actions taken from debugging provenance queries improved the quality of results and time spent to complete all computations in bioinformatics scenarios. Similar results were obtained in different domains such as deep water oil exploitation (Ogasawara et al. 2011), numerical reduced order models (Dias et al. 2011) and text mining (Dias et al. 2013).

Data storage and access from thousands workflow executions are also important issues. When workflow execution and provenance data are in the same database, queries are simpler to formulate and a lot of data redundancy is avoided. However, access to this database may become a bottleneck. Using well-known parallel relational database techniques and new lighter SQL systems may leverage this. This integrated provenance database only has metadata for one specific execution. When the workflow finishes its execution, this database may be loaded into a larger provenance warehouse. We believe that database technology has an unexplored potential to further improve existing solutions for workflow debugging in parallel large data-flow executions.

References

Abouelhoda, M., Issa, S., Ghanem, M., (2012), "Tavaxy: Integrating Taverna and Galaxy workflows with cloud computing support", *BMC Bioinformatics*, v. 13, p. 77.

Ailamaki, A., Kantere, V., Dash, D., (2010), "Managing scientific data", *Communications of the ACM*, v. 53, n. 6 (Jun.), p. 68–78.

Costa, F., Silva, V., de Oliveira, D., Ocaña, K., Ogasawara, E., Dias, J., Mattoso, M., (2013), "Capturing and Querying Workflow Runtime Provenance with PROV: A Practical Approach". In: *EDBT/ICDT '13 Workshops*, p. 282–289, New York, NY, USA.

Dias, J., Ogasawara, E., Oliveira, D., Porto, F., Coutinho, A., Mattoso, M., (2011), "Supporting Dynamic Parameter Sweep in Adaptive and User-Steered Workflow". In: 6th WORKS, p. 31–36, Seattle, WA, USA.

Dias, J., Ogasawara, E., de Oliveira, D., Porto, F., Valduriez, P., Mattoso, M., (2013), "Algebraic dataflows for big data analysis". In: *2013 IEEE International Conference on Big Data2013 IEEE International Conference on Big Data*, p. 150–155

Freire, J., Koop, D., Santos, E., Silva, C. T., (2008), "Provenance for Computational Tasks: A Survey", *Computing in Science and Engineering, v.10*, n. 3, p. 11–21.

Gadelha, L. M. R., Wilde, M., Mattoso, M., Foster, I., (2012), "MTCProv: a practical provenance query framework for many-task scientific computing", *Distributed and Parallel Databases*, v. 30, n. 5-6 (Oct.), p. 351–370.

Gonçalves, J. C. A. R., Oliveira, D., Ocaña, K. A. C. S., Ogasawara, E., Mattoso, M. (2012) "Using Domain-Specific Data to Enhance Scientific Workflow Steering Queries", In: IPAW 2012, p. 152-167

Gunter, D., Deelman, E., Samak, T., Brooks, C. H., Goode, M., Juve, G., Mehta, G., Moraes, P., Silva, F., et al., (2011), "Online workflow management and performance analysis with Stampede". In: *Network and Service Management (CNSM), 2011 7th International Conference on*, p. 1 –10

Hey, T., Tansley, S., Tolle, K., (2009), *The Fourth Paradigm: Data-Intensive Scientific Discovery*. Microsoft Research.

Ikeda, R., Cho, J., Fang, C., Salihoglu, S., Torikai, S., Widom, J., (2012), "Provenance-Based Debugging and Drill-Down in Data-Oriented Workflows". IEEE 28th International Conference on Data Engineering (ICDE), p. 1249–1252

Lee, K., Paton, N. W., Sakellariou, R., Deelman, E., Fernandes, A. A. A., Mehta, G., (2008), "Adaptive Workflow Processing and Execution in Pegasus". In: *3rd International Conference on Grid and Pervasive Computing*, p. 99–106, Kunming, China.

Mattoso, M., Ocaña, K., Horta, F., Dias, J., Ogasawara, E., Silva, V., de Oliveira, D., Costa, F., Araújo, I., (2013), "User-steering of HPC workflows: state-of-the-art and future directions". In: *2nd SWEET ACM SIGMOD Workshop*, p. 1–6, New York, NY, USA.

Ocaña, K. A. C. S., Oliveira, D., Ogasawara, E., Dávila, A. M. R., Lima, A. A. B., Mattoso, M., (2011), "SciPhy: A Cloud-Based Workflow for Phylogenetic Analysis of Drug Targets in Protozoan Genomes". In: *BSB*, p. 66–70, Berlin, Heidelberg.

Ocaña, K., Benza, S., Oliveira, D., Dias, J., Mattoso, M., (2014), "Exploring Large Scale Receptor-Ligand Pairs in Molecular Docking Workflows in HPC Clouds". In: *13th IEEE International Workshop on High Performance Computational Biology (HiComb 2014)*, IPDPS, Phoenix, Arizona, USA, p. 536-545.

Ogasawara, E., Dias, J., Oliveira, D., Porto, F., Valduriez, P., Mattoso, M., (2011), "An Algebraic Approach for Data-Centric Scientific Workflows", *PVLDB Endowment*, v. 4, n. 12, p. 1328–1339.

Oliveira, D., Ocaña, K. A. C. S., Ogasawara, E., Dias, J., Gonçalves, J., Baião, F., Mattoso, M., (2013), "Performance evaluation of parallel strategies in public clouds: A study with phylogenomic workflows", *Future Generation Computer Systems*, v. 29, n. 7 (Sep.), p. 1816–1825.

Oliveira, D., Ogasawara, E., Baião, F., Mattoso, M., (2010), "SciCumulus: A Lightweight Cloud Middleware to Explore Many Task Computing Paradigm in Scientific Workflows". In: *3rd International Conference on Cloud Computing*, p. 378–385, Washington, DC, USA.

Pennisi, E., (2011), "Will Computers Crash Genomics?", *Science*, v. 331, n. 6018 (Feb.), p. 666–668.

Vahi, K., Harvey, I., Samak, T., Gunter, D., Evans, K., Rogers, D., Taylor, I., Goode, M., Silva, F., et al., (2013), "A Case Study into Using Common Real-Time Workflow Monitoring Infrastructure for Scientific Workflows", *Journal of Grid Computing*, v. 11, n. 3 (Sep.), p. 381–406.

Wozniak, J. M., Armstrong, T. G., Maheshwari, K., Lusk, E. L., Katz, D. S., Wilde, M., Foster, I. T., (2012), "Turbine: a distributed-memory dataflow engine for extreme-scale many-task applications", p. 1–12