Integrating Domain-data Steering with Code-profiling Tools to Debug Data-intensive Workflows

Vítor Silva[§], Leonardo Neves[§1], Renan Souza[§◊], Alvaro Coutinho[§],
Daniel de Oliveira[★], Marta Mattoso[§]

[§]Federal University of Rio de Janeiro (COPPE/UFRJ), Rio de Janeiro, Brazil

[★]Computing Institute, Fluminense Federal University (IC/UFF), Niterói, Brazil

[◊]IBM Research Brazil

{silva, renanfs, marta}@cos.ufrj.br, lrmneves@poli.ufrj.br,
alvaro@nacad.ufrj.br, danielcmo@ic.uff.br

ABSTRACT

Computer simulations may be composed of several scientific programs chained in a coherent flow and executed in High Performance Computing environments. These executions may present execution anomalies associated to the data that flows in parallel among programs. Several parallel code-profiling tools already support performance analysis, such as Tuning and Analysis Utilities (TAU) or provide fine-grain performance statistics such as the System Activity Report (SAR). These tools are effective for code profiling, but do not associate their results to dataflow analysis neither are connected to the concept of data-intensive workflows. Such analysis is fundamental and may be performed by capturing execution data enriched with fine-grain domain data during the long-term run of a computer simulation. In this paper, we propose a monitoring data capture approach as a component that couples code-profiling tools to metadata from workflow executions. The goal is to profile and debug parallel executions of workflows through queries to a database that integrates performance, resource consumption, provenance, and domain data from simulation programs flow at runtime. We show how querying this database enables domain-aware runtime steering of performance anomalies by using the astronomy Montage workflow. We show that the overhead introduced by our approach was 0.6% of the total workflow execution time.

Keywords

Performance analysis; debugging; scientific workflow; provenance.

1. INTRODUCTION

Oden *et al.* [1] define models as mathematical constructions based on physical principles attempting to characterize abstractions of the reality. Computational simulations make possible to use mathematical models with tremendous complexity for solving problems of relevance in science and engineering. Commonly, these simulations involve the execution of complex programs chained in a coherent flow. Each program in the flow represents a computational model execution with a data dependency from a previous simulation program in the flow. This flow of programs can be modeled as a scientific workflow (for the sake of simplicity henceforth named as workflow) [2].

A workflow is an abstraction that defines a set of activities and a dataflow among them [2]. Each activity is associated to a simulation program, which is responsible for the consumption of an input dataset and the production of an output dataset. Many workflows process a large volume of data, requiring the effective usage of High Performance Computing (HPC) or High-throughput Computing (HTC) environments allied to parallelization techniques such as data parallelism or parameter sweep [3].

To support the modeling and execution of workflows in those environments, standalone parallel Scientific Workflow Management Systems (SWfMS) were developed, such as Swift/T [4], Pegasus [5] and Chiron [6], or SWfMS embedded in Science gateways such as WorkWays [7]. To foster data parallelism, the activities of workflows can be instantiated as tasks for each input data, known as activations [6] (we are going to use the term *activation* consistently throughout this paper). Each activation executes a specific program or computational service in parallel, consuming a set of parameter values and input data and producing output data. Besides the activations, parallel SWfMS control the data dependencies among activities. It is worth mentioning that the dependency management of this dataflow and provenance support are some of the advantages of SWfMS in relation to executing workflows using Python scripts or Spark [8].

These SWfMS consume large amounts of resources, however, as the workflow becomes increasingly data-intensive (measured by the number of activations or the volume of processed data), they tend to execute for days or even weeks depending on the amount of input data and the availability of computational resources. Thus, it is fundamental for users to be aware of the execution status and to analyze partial data results to check if the current execution complies with some pre-defined quality and performance criteria. Based on data steering [9], users may decide if they have to interfere in the execution (also known as dynamic workflows [10]). Data steering at runtime allows for users to analyze partial results and to potentially identify failures and problems as early as they happen, as highlighted by Ailamaki *et al.* [11] as *online* analysis of scientific workflows. In previous papers, we have experienced the benefits of domain-data steering, when running workflows in different domains from engineering [12] to astronomy [13] and biology [14]. However, analyzing a high-level execution data associated to domain data is not enough to debug and trace performance anomalies. Considering the efficiency and high quality support of code-profiling tools we present an approach to couple these tools to domain-data steering in workflows.

It is far from trivial to monitor and steer the performance with the resource consumption related to domain data during the parallel execution of workflows in distributed environments [15]. For

[*]Leonardo Neves is currently at Language Technologies Institute, Carnegie Mellon University, Pittsburgh, PA, USA.

example, the same workflow (the same conceptual specification) can be executed in several physical configurations, each one with its own characteristics and storing different portions of data or executing alternative programs that are part of the same workflow. In addition, in projected exascale-computing systems, factors such component, failures, network latency and bandwidths, and performance variability, will make it increasingly more difficult to execute and manage complex scientific workflows. New approaches for performance and resource consumption monitoring and analysis are important to overcome programming complexities envisioned in these exascale environments.

Users need to relate performance and resource consumption information with domain-specific data to plan actions. For example, the execution of simulation programs with several combinations of parameter values correspond to the production of many data files, whose contents present relevant domain data to the result analysis. Despite the challenges to find those several data files, users have to develop *ad-hoc* programs to access and extract the contents of these files (often binary or specific formats) to analyze the workflow results. These files may have a huge volume of data. Another challenge on the exploratory analysis of these raw data files, which also happens for monitoring analysis (*i.e.*, query to the domain and performance data), is to relate domain-specific data from files that are manipulated by different simulation programs (*i.e.*, activities).

Most of the parallel SWfMS have addressed the need of performance and resource consumption monitoring facilities by adding new components in their workflow engines, or loading data into databases (at runtime or after the workflow execution) to be further queried by users [9]. Approaches such as STAMPEDE [16], which has been coupled to Pegasus, are also able to monitor the execution of workflows in HPC environments at runtime. These monitoring mechanisms are able to provide information such as the execution time of each activity or activation of the workflow, how many activations were executed, and resource usage. This type of information is very useful to help users to debug an execution or to understand the performance of a workflow in a specific environment. However, this information is at coarse grain and due to that, users miss an opportunity to understand the behavior of the data derivation (*i.e.*, dataflow path) associated to the performance and resource consumption.

To address low level execution information, there are code-profiling tools that support debugging and profiling of HPC scientific applications, such as Tuning and Analysis Utilities (TAU) [17]. TAU instruments the application code to capture performance data and invokes ParaProf for presenting these data, for instance, 3D visualizations. Other tools, such as System Activity Report (SAR)[2], provide system statistics from time to time, but disconnected from the workflow execution data.

When performance and resource consumption data are not related to fine-grained domain data (*i.e.* data value within data files), the user may not see that a certain data value from a huge file is presenting an anomalous behavior. By knowing that an activity is taking too long, users can investigate further input and output data for that particular activation. These scenarios emphasize the importance of exploratory analysis of multiple related raw data files [13].

Our previous papers have shown the importance of users for monitoring information related to the workflow execution, its performance information, resource consumption information, dataflow, and domain-specific data to support workflow steering and put the human-in-the-loop [18]. We also showed the importance of storing those data in a relational provenance database and following a provenance data model that is compliant to W3C PROV [19]. We present PROV-Df [13], a provenance data model that manages dataflow generation during the execution of scientific workflows.

Computational simulations workflows usually involve several users with different skills. Broadly speaking, we can consider three types of users in the process: (i) users\developers who have expertise on using domain specific programs; (ii) domain scientists who analyze the final results and (iii) computer science experts. Existing solutions should support these types of users. The domain scientist plays a key role in the debugging process and cannot be forgotten. Developers that designed the workflow are the only ones able to identify particular choices of parameters or programs that are not working. They are able to decide whether an anomaly is in fact an error to be debugged/fixed or just an outlier activation that is consuming more time/memory/cpu than usual.

In this paper, we present a database-oriented approach that is able to extract and represent fine-grained performance with resource consumption data associated to workflow information, provenance and domain-specific data all into a single database managed by a Relational Database Management System (RDBMS) at runtime. In [13] and [12], we showed how domain and provenance data associated to execution data are able to improve steering, debugging and workflow execution time. However, execution data was limited to performance data captured by SWfMS. Thus, users still had to explore TAU or other tools to improve debugging, while having a hard time to associate debugging tools to the enriched provenance database.

We also develop a component for capturing performance and resource consumption metrics that is designed on top of TAU and SAR tools, which we named as *PerfMetricEval*. We coupled *PerfMetricEval* to Chiron SWfMS to present a relational database that integrates provenance, domain-specific, performance, and resource consumption data. This integrated database enables users to perform their analysis by submitting simple SQL queries to the RDBMS. Therefore, we can enumerate our contributions in this paper:

- Development of *PerfMetricEval* component to capture performance and resource consumption data using TAU and SAR profiling tools;
- A specialization of PROV-Df to include typical data from the code-profiling tools;
- Integration of *PerfMetricEval* API with an existing SWfMS. In this case, we use Chiron SWfMS.

This paper is organized in five sections. Section II discusses related work. Section III describes our approach for performance and resource consumption monitoring; the extended provenance database schema that integrates provenance, domain, performance and resource consumption data, all together; and the integration between *PerfMetricEval* with Chiron SWfMS. Section IV shows the evaluation of the proposed approach using the Montage workflow in a cluster environment and we conclude the paper in Section V.

2. RELATED WORK

There are several SWfMS that provide monitoring and performance analysis mechanisms within their engines. ASKALON [20], Swift/T, Pegasus (kickstart tool [21]), Makeflow [22] and Chiron already provide monitoring mechanisms for users to follow the execution of the workflow and to analyze its behavior. Swift/T and Pegasus

[2] http://pubs.opengroup.org/onlinepubs/7908799/xsh/sysstat.h.html

provide interfaces to follow the execution while Chiron is based on database queries performed at runtime for monitoring. Swift/T and Pegasus provide information about the amount of activations executed, the execution time of each activation and the resources used. Specifically, Pegasus SWfMS uses kickstart tool to perform performance analyses that can also be associated to captured provenance data. Makeflow is a workflow approach that enables performance monitoring and debugging on HPC environments, such as application elapsed time and the volume of data processed in a specific time interval. Chiron provides information about the amount of activations, their execution times, and domain data in the same database, but does not provide performance metrics neither resource consumption data. STAMPEDE [16] is a SWfMS-independent solution that provides a common model for workflow monitoring. It provides a large set of metrics for monitoring, but does not consider domain-specific data, which is valuable for users.

WorkWays [7] and FireWorks [24] enable users to monitor the status and the elapsed time of tasks, and correlate those data to domain-specific data. However, those performance data limit some analyses in HPC environments, such as the identification of performance bottlenecks associated to the memory usage or I/O operations in the disks.

There are other approaches that provide detailed performance and resource consumption information for general applications (*i.e.* disconnected from the workflow concept). Tuning and Analysis Utilities (TAU) [17] is a profiling tool that gathers performance information and visualize it on interactive graphs using ParaProf. TAU gathers performance information by instrumenting functions, methods, basic blocks, and statements as well as event-based sampling. To use TAU in workflows, it is required to instrument both the applications and the workflow engine to collect performance data. With the same purpose of TAU, DARSHAN is a resource consumption profiler that monitors I/O operations in applications with a non-intrusive solution. DARSHAN could also be coupled to SWfMS.

System Activity Report (SAR) is a Linux monitor command that informs system loads, including CPU activity, memory usage statistics, device load, *etc*. The statistics provided by SAR are fine-grained, but this approach is disconnected from the workflow concept. To use SAR in SWfMS, one should couple it to the workflow engine or call it within the program that is invoked. We have used SAR to help on other workflow applications, but associating SAR information to provenance and domain-specific data is far from trivial. Similarly to SAR, CCTools[3] presents a resource monitor tool for gathering performance data during the execution of applications. CCTools presents tools to enable visualization of performance data, such as the memory usage, I/O and % of cpu usage. Ganglia [23] is a distributed monitoring system for distributed infrastructures such as clusters and grids. In Ganglia, users are able to view the CPU usage over the past hour, day, week, month, or year. Ganglia also presents similar visualizations for memory, disk usage, network statistics, number of running processes, *etc*. However, Ganglia usage is similar to SAR's and CCTools', and thus they all present the same difficulties to associate performance data with provenance and domain-specific data.

Analyzing related work, we observe that existing approaches do provide valuable information for computer science experts to debug an execution or to understand the performance of a workflow or a general scientific application in a specific HPC environment. However, this information is far from the application data, so users miss important opportunities to understand the behavior of data derivation based on the resource consumption. When resource consumption data are not related to domain data, it may be hard to find which specific data value is presenting an anomalous execution behavior and act on this.

3. THE PERFMETRICEVAL COMPONENT

Several existing tools already support debugging and profiling of HPC scientific applications for computer experts, such as TAU, which instruments the application code to capture performance data and to present these data using a graphical representation. However, domain users need to analyze performance and resource consumption together with the domain-specific data, as well as to be aware of all data transformations that have occurred in the workflow parallel execution. In this section we show how TAU, SAR and other tools may be integrated to a PROV database of a SWfMS. To integrate TAU, SAR and the provenance database, we have extended a W3C PROV compliant provenance database schema with performance and resource consumption information.

3.1. Performance and Resource Consumption Metrics

It is well-known that performance metrics such as speedup and efficiency are extensively used in HPC community to measure the performance of parallel applications, including parallel workflows. Besides the speedup, it is also important to measure the resource consumption in the HPC environment. Those metrics are fundamental to get a general idea of the gain in application performance *versus* the amount of resources to achieve this gain. However, for further examination of performance and resource consumption, several additional metrics are required. In this paper we consider metrics such as total elapsed time (that can be decomposed to identify bottlenecks related to the computer simulation; for example, communication bottleneck), CPU usage, memory consumption and I/O, and transfer rates statistics to be captured and stored in the provenance database.

In this paper, we decomposed total workflow elapsed time, which corresponds to the workflow wall-clock time, into three different metrics: useful computing time (time needed for executing a specific activation – *T_comp*), communication time (time needed to perform communication between processes/machines – *T_comm*) and time taken to access the provenance database (*T_prov*), thus $T\_wf = T\_comp + T\_comm + T\_prov$.

For the CPU usage, we consider the cumulative runtime CPU usage of all machines involved in the execution of the workflow. The CPU usage can be decomposed into the percentage of CPU utilization that occurred while executing the application (*usr*), the percentage of CPU utilization that occurred while executing at the system level (*sys*), the percentage of time that the CPU was idle during which the system had an outstanding disk I/O request (*iowait*) and the percentage of time that the CPU was idle and the system did not have an outstanding disk I/O request (*idle*).

For memory consumption, we consider the amount of memory used during the execution of each activation of the workflow. The memory consumption metric can be decomposed into the amount of free memory available (*kbmemfree*), the amount of used memory in kilobytes (*kbmemused*), the percentage of memory used (*memused*), the amount of memory used as buffers by the kernel (*kbbuffers*), the amount of memory used for cache data by the kernel (*kbcached*) and

3 http://ccl.cse.nd.edu/software

the estimated amount (by OS) of memory needed for the current workload (*kbcommit*).

For I/O and transfer performance, we consider I/O and transfer rate statistics that can be decomposed into two metrics. The first metric is the total amount of data read from the drive in blocks per second (*blk_read/s*) and the second metric is the total amount of data written to the drive in blocks per second (*blk_write/s*). In our experiments we used Linux Cent OS 5.5 where blocks have a size of 512 bytes.

These are the currently supported metrics. All those statistics may be provided by systems such as Ganglia or SAR. *PerfMetricEval* aims to capture those performance and resource consumption metrics using both TAU and SAR. Using TAU, we capture the elapsed time of computing, communication, and provenance operations. However, TAU does not provide memory, cpu and I/O statistics. Thus, we capture fine-grained performance data, such as the percentage of CPU utilization and the amount of memory used during the execution of each activation of the workflow, using SAR. In Section 3.2 we present an extension to the PROV database schema to represent the captured performance and resource consumption data using *PerfMetricEval* component further described in Section 3.3.

3.2. A PROV-Compliant Database Schema

In our provenance database schema, provenance and domain-specific data are represented as tables. Each of these tables references other tables that represent activations performed for a given computer simulation. The provenance database follows the PROV-Df model [13] that is compliant to the W3C PROV recommendation. Thus, it is possible to relate information of the status of the workflow to domain-specific data. However, the database does not contain performance and resource consumption information associated to provenance and domain-specific data. Figure 1 presents an excerpt of the provenance database, showing both the entities that represent retrospective provenance (*i.e.* data related to the workflow execution in white) and performance and resource consumption information (in light gray), which is the focus of this paper. PROV-Df compliance with W3C PROV-DM was validated using ProvToolbox[4] to generate files according to the PROV-DM specification (in PROV-N[5], JSON[6], XML[7], and DOT[8] file formats) and ProvValidator[9] web service to analyze these generated files.

The *eWorkflow* table stores information of the current execution of the workflow and *eActivity* table stores provenance data related to each activity (which is associated with an application) execution, such as start and end execution times. The *eFile* table stores information (*e.g.*, size, directory, and name) of the files consumed and produced in the execution for each activation. The *eMachine* table stores details about the machines of the HPC environment where activations are executed. The *eActivation* table is important for monitoring the workflow execution performance since it stores information of the status of each activation such as whether it has completed successfully or faulty (*status* and *exitstatus*), or which errors were launched by the program (*terr*), the number of execution attempts that failed (*failure_tries*), and in which node (*machineid*) and processor (*processor*) the activation is being performed. Thus, it can monitor the workflow status with a high level of detail using provenance data and domain-specific data that was previously extracted from produced raw data files [13].

In Figure 1 the table *InputFitPlane* (in dark gray) represents the domain-specific data from Montage workflow (which is detailed in Section 4). One of the main advantages of using Chiron's database is that there is a native relationship between the execution data and domain-specific data. The table *InputFitPlane* is an example of domain table with input data for a particular activity. Each activation consumes a tuple (or a set of tuples, depending on the activity). Due to the relationship with *eActivation* table, one can associate this tuple (or set of tuples) with a tuple in the domain table. Consequently, we can thoroughly analyze the execution at runtime through simple SQL queries.

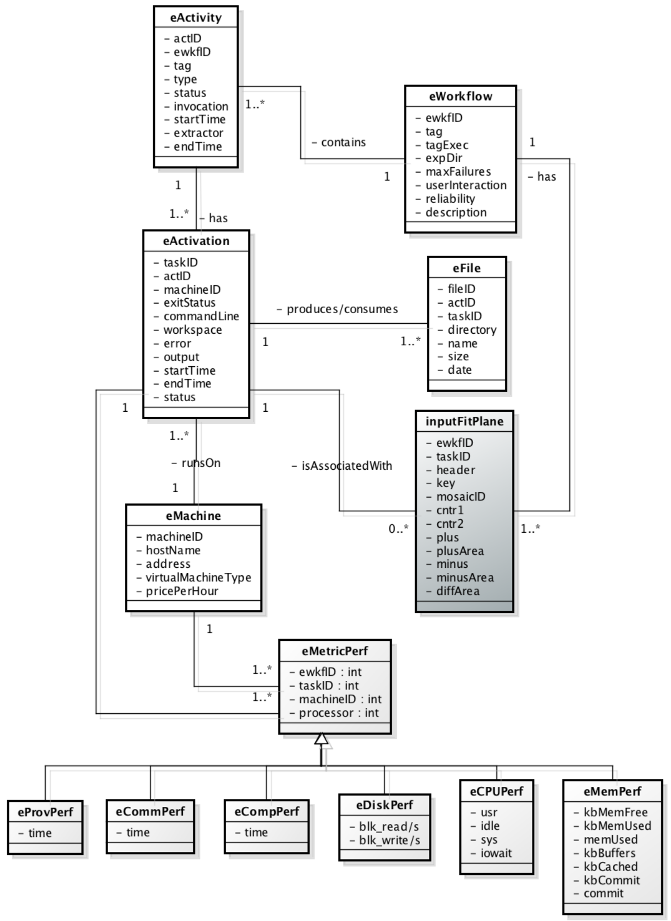

Figure 1. An excerpt of the PROV database schema

Nevertheless, the aforementioned tables of the model do not consider performance and resource consumption information. To store this type of information and relate it to provenance and domain data, we created six new tables in the provenance database (in light gray): *eCompPerf, eProvPerf, eCommPerf*, *eCPUPerf*, *eMemPerf* and *eDiskPerf*. Table *eCompPerf* stores values related to the actual time needed to execute the program associated to the activity in a specific machine. The *eProvPerf* table stores values related to the time required to store provenance data of each activation in the database and *eCommPerf* stores values associated to the time spent in communication for each activation.

The *eCPUPerf* table stores information about the CPU usage during each activation. It contains the attributes *idle* (percentage of idle CPU

[4] http://lucmoreau.github.io/ProvToolbox
[5] https://s3.amazonaws.com/works2016/chiron.provn
[6] https://s3.amazonaws.com/works2016/chiron.json
[7] https://s3.amazonaws.com/works2016/chiron.xml
[8] https://s3.amazonaws.com/works2016/chiron.dot
[9] https://provenance.ecs.soton.ac.uk/validator/view/validator.html

time), *usr* (percentage of time the CPU was running user processes), *sys* (percentage of time the CPU was running system processes) and *iowait* (percentage of time the CPU was waiting for a disk I/O request). The *eMemPerf* table stores information related to the memory utilization during the execution of a specific activation. It contains the attributes *kbmemfree* (amount of free memory available in kilobytes), *kbmemused* (amount of used memory available in kilobytes - it does not take into account memory used by the kernel itself), *memused* (percentage of memory used), *kbbuffers* (amount of memory used as buffers by the kernel in kilobytes), *kbcached* (amount of memory used to cache data by the kernel in kilobytes), *kbcommit* (amount of memory in kilobytes needed for current workload - this is an estimation of how much memory is needed to guarantee that there never is out of memory) and *commit* (percentage of memory needed for the current workload in relation to the total amount of RAM memory and swap memory). The value of commit attribute may be greater than 100% because the kernel usually overcommits memory in several Linux distributions. The *eDiskPerf* table stores information related to the disk usage statistics during an activation execution. It contains the attributes *blk_read/s* (the total amount of data read from the drive in blocks per second) and *blk_write/s* (total amount of data written to the drive in blocks per second).

3.3. Integrating PerfMetricEval to Chiron

To capture performance and resource consumption metrics using TAU and SAR, we developed a component, named as *PerfMetricEval*. The integration of Chiron with *PerfMetricEval* is based on inserting an invocation of *PerfMetricEval* component (after the execution of each activation) to gather fine-grain performance information from TAU and SAR, insert this data in the provenance database and then convert it into TAU files (profiles for each computing node, *e.g. profile.0.x.0* for node *x*). The generated TAU files serve as input for TAU to plot, for instance, 3D graphs using ParaProf [17]. In this first version of Chiron coupled to *PerfMetricEval*, we may plot graphs considering all metrics described in Section 3.1, although new metrics can be added without much effort. An overview of *PerfMetricEval* execution flow is presented in Figure 2. For reproducibility reasons, a version of Chiron coupled to *PerfMetricEval* can be downloaded at https://s3.amazonaws.com/works2016/Chiron-PerfMetricEval+.zip. A simple README is provided to show how to configure a database and invoke Chiron + *PerfMetricEval.*

After the execution of each activation, Chiron invokes the *PerfMetricEval* component that identifies the elapsed time of the activation and invokes SAR to gather resource consumption information related to the corresponding activation. For each metric presented in Section 3.1, it is created a new file with the name *<IDActivation>_<Metric name>.metric* and stored in the workspace of the workflow (the local directory where the activation is executed). For example, activation with ID 435 would produce three files: *435_cpu.metric*, *435_mem.metric* and *435_disk.metric*. Each file contains the statistics provided by SAR. By creating those files, Chiron is able to parse them, extract performance information, and asynchronously load it into the provenance database. Thus, it does not impact the workflow performance (*i.e.* we do not have to wait until the end of extraction of consumption information to start the next activation of the flow). It is also a distributed approach, since Chiron is distributed and runs *PerfMetricEval* in all machines of the HPC environment concurrently. After that, Chiron integrates those performance data gathered in a distributed manner. All data is loaded in the database and associated with the ID of the activation (*taskID* in Figure 1).

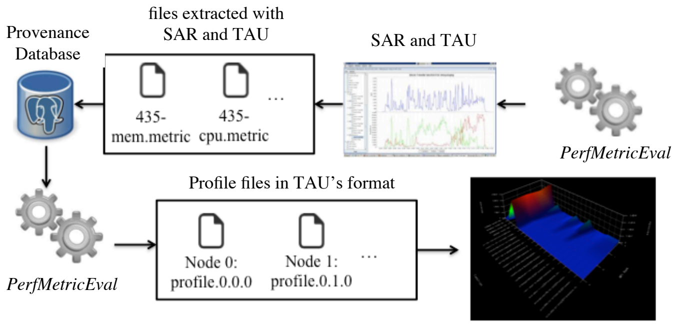

Figure 2. **The *PerfMetricEval* execution flow**

After loading the performance and resource consumption data into the same database, the *PerfMetricEval* component also provides a feature to query the data relevant to the user-defined parameters for performance and resource consumption analysis by executing standard SQL queries made by users. After querying the database and gathering the results, the *PerfMetricEval* provides a component to generate a file in TAU format to profile of the execution. The TAU visualization tool (paraprof command) graphically displays the generated files. Since the used format is compatible with the code-profiling tool, it enables the creation of images as bar graphs, 3D meshes and scatter charts, all interactive and with customization capabilities inherent to TAU. This integration of *PerfMetricEval* and Chiron enables in-depth analysis graphs generated within seconds and support decision-making by users at runtime.

4. EXPERIMENTAL EVALUATION WITH MONTAGE WORKFLOW

This section presents the case study used to evaluate the proposed approach. We use the well-known scientific workflow Montage from the astronomy domain as the case study. Firstly, we describe how we modeled the Montage workflow and then the analyses performed.

4.1. Montage Workflow

We modeled the Montage workflow (Figure 3) using an astronomy toolkit for assembling astronomical images into custom mosaics using a suitable format for large scale data processing of the sky [25]. This toolkit comprises a set of components that provide astronomy image mosaic services to build mosaics in the Flexible Image Transport System (FITS) file format [26]. FITS format respects the common astronomy coordinate system, arbitrary image sizes and rotations, and all World Coordinate System (WCS) map projections. We modeled the scientific workflow to use different astronomical images in order to blend them into custom mosaics, considering the necessary geometric transformations. For more details about Montage, please refer to Jacob *et al.* [25].

4.2. Performance Results

In this section, we present how users are able to analyze their workflows using provenance, domain-specific, performance and resource consumption data, all together. We present how users are able to extract information from the provenance database and the results from a (small-scale) Montage workflow execution. The results presented in this section refers to a workflow execution that lasted approximately 17 hours in a SGI Altix ICE 8200 at NACAD/COPPE/UFRJ with four machines 2x Quad Core Intel Xeon X5355 2.66 GHz (32 cores).

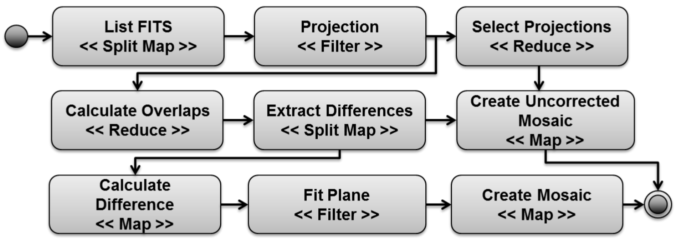
List FITS << Split Map >>
Projection << Filter >>
Select Projections << Reduce >>
Calculate Overlaps << Reduce >>
Extract Differences << Split Map >>
Create Uncorrected Mosaic << Map >>
Calculate Difference << Map >>
Fit Plane << Filter >>
Create Mosaic << Map >>

Figure 3. Astronomy workflow using algebraic operators

The Montage execution consumed 1,585 input file images, which produced approximately 17,500 activations that were executed in parallel. After each one of these activations, the performance and resource consumption information were captured and loaded into the database. Next we illustrate with some useful analyses the proposed approach.

A basic analysis assessment is the workflow computing time in each machine. One simple SQL query can sum the actual computing time, the time spent with communication and the time needed for storing provenance for each activation and group by each used machine. This query is able to extract data for the three derived metrics: computation time (*T_comp*), communication time (*T_comm*) and provenance (*T_prov*) for each machine. We present an excerpt of the extracted data after the query execution. The results are presented in Table 1, while a visualization is shown in Figure 4. In addition, using the data in Table 1 and the metrics described in Section III, we have the total workflow execution time (*T_wf)* as 1,026 minutes ≈ 17 hours.

Table 1. Computing time in each machine in minutes

	T_comp	***T_comm***	***T_prov***	***T_wf***
Machine0	533.33	96.33	24.02	653.68
Machine1	923.33	30.67	0.00	954.00
Machine2	993.22	33.00	0.00	1,026.22
Machine3	961.67	32.67	0.00	994.34

Analyzing Figure 4 we can also state that only machine0 loads provenance data in the database. It is due to the architectural characteristics of Chiron, where only the master node is responsible for storing provenance data in the provenance database. This was an architectural choice for Chiron, since the slave nodes can process new activations without being locked by the provenance management. We can also state that Machine0 is the one that presents the highest communication overhead, because it integrates provenance data storage and all machines send provenance data to it using messages, increasing the communication cost for Machine0.

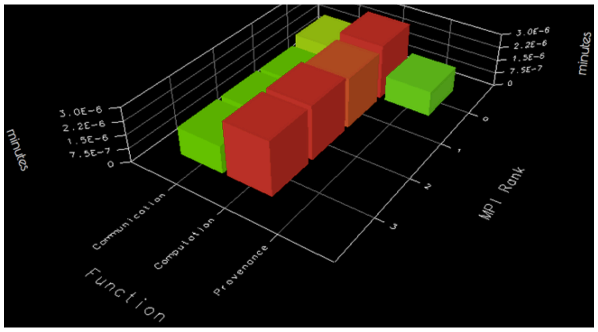

Figure 4. The computing time of each machine

Another important analysis considers activity average resource consumption per machine. Besides, users usually need to relate the resource consumption and the domain-specific data. Domain scientists commonly have a fairly good estimation of the needed execution time for a specific activation (based on their experience and previous executions, all registered in the provenance database). Using Chiron + *PerfMetricEval* they are able to check if the real activation execution time or resource consumption meets the estimate. If the real execution time is considerably higher than the estimate they are able to identify an anomalous behavior. For example, it is well known for Montage users that the image region of interest can impact on the performance and resource consumption of workflow activations. Thus, users often need to explore the behavior of a specific subset of input data. In this case, the image region of interest can be defined by setting the domain attributes CRPIX1 and CRPIX2 (which values are also loaded from data files to the provenance database) that represent the pixels that define the image interest region. Thus, one query can retrieve the average memory consumption, the average memory used, the average CPU usage, the average CPU usage for the operational system, the average amount of disk blocks read/write, *etc*. All this data is associated with the domain attributes CRPIX1 and CRPIX2. With the result of this query, the domain user can monitor the performance of activations for building the mosaic (*Create Mosaic*), limited to only one specific image region to check if there is an anomaly in the execution. The data extracted from this query allowed us to generate graphs of Figure 5, Figure 6, Figure 7, and Figure 8.

In Figure 5 and Figure 6 we see the CPU usage per machine when executing activations where $100 < crpix1 < 150$ and $50 < crpix2 < 80$. Since Chiron considers the CPU usage on its scheduling algorithm we can state that there is a load balancing among the machines from the CPU usage perspective, *i.e.*, all machines present an equivalent CPU consumption considering both idle (in light blue), iowait (dark blue), sys (green) and user metrics (red). However, the same behavior is not found when we consider both memory usage statistics and disk usage statistics.

In Figure 7 we can visualize the number of blocks read per second (blue) and blocks written per second (green for machine 1, red for machines 0 and 2 and yellow for machine 3). We can observe that machine 0 and 2 present a larger value for the blocks written in disk per second. It means that the activations executed on these machines are more I/O intensive than the others scheduled for machines 1 and 3. One possible reason is that the scheduling algorithm of Chiron does not consider the amount of data to write on disk so I/O intensive activations can be placed in the same machine. Another reason is that parameters set for the workflow activities are not tuned, which may increase the amount of data written. By being aware of this information, we can tune parameters and find a balanced execution. It can be also used for helping Chiron developers to enhance the scheduling algorithm. For example, the average number of blocks written per second on machine 0 is 7,565.33 while machine 1 is 3,059.00, machine 2 is 8,078.77 and machine 3 is 5,743.24.

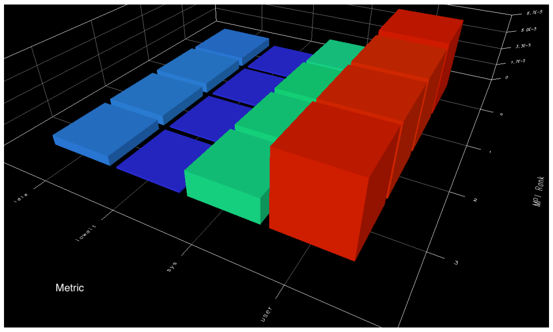

Figure 5. The CPU statistics for activations executed

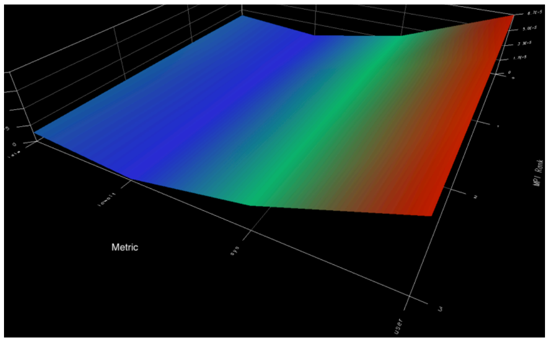

Figure 6. Another view of The CPU statistics for activations executed in each machine

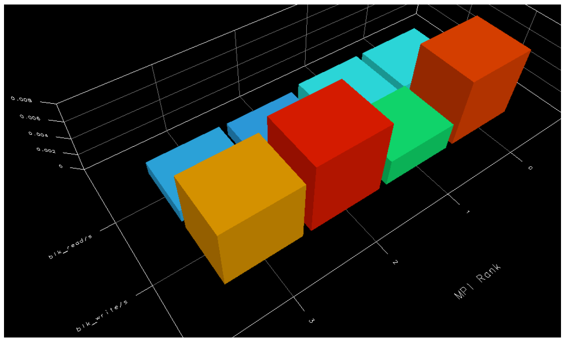

Figure 7. The disk statistics for activations executed in each machine

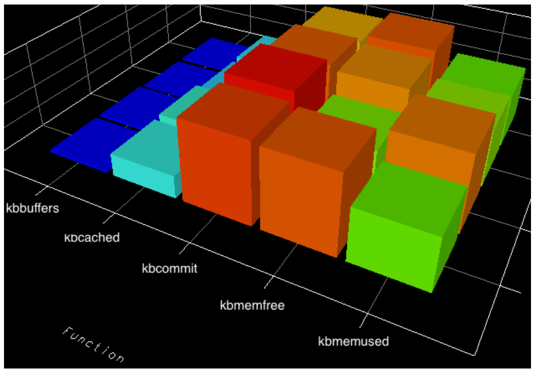

Figure 8. The memory consumption statistics for activations executed in each machine

A similar behavior happens when we study the memory consumption statistics. We observe in Figure 8 that machines 0 and 2 present a higher memory consumption than the other machines. Similar to the disk analysis, Chiron does not consider the amount of memory used to execute the activation while scheduling the activations, thus unbalancing (from the memory consumption perspective) may happen. For example, the amount of memory used (in Kbps) and the percentage of memory used for machine 0 is 5,678,976 and 67.7% while in machine 1 is 3,617,587 and 43.1%, machine 2 is 4,875,878 and 58.1% and machine 3 is 3,323,985 and 39.6%. These results indicate that changes in the scheduling algorithm may be needed, but it is also useful for better fine-tune the workflow parameters since they are now aware of the characteristics.

We also evaluated the overhead imposed by the proposed approach for monitoring workflows using fine-grained domain data. We measured the time needed to extract domain data from files, to execute SAR and extract resource consumption statistics and the time needed to load all data in the provenance database. In Figure 9 we present the methods in Chiron code that are invoked to execute those functions. The provenance and extraction methods do not present a negligible time in terms of absolute values (some minutes in total). For example, the methods *loadReadyActivation* (represented in the red bar), *checkIfAllActivationsFinished* (in yellow), *extractSARMetrics* (3 bars in blue: CPU, Disk and Memory) and *loadActivation* (in green bar) are those with the largest bottlenecks.

This result is particularly important for Chiron and other SWfMS developers, since it shows that concentrating optimization efforts on these methods is an important event. Thus, the time needed to perform all functions using Montage workflow with the aforementioned input dataset is approximately 7.73 minutes in total, which corresponds to 0.6% of the total workflow execution time. This fact shows that Chiron + *PerfMetricEval* approach, based on performance data queries, does not add significant overhead in the workflow execution time.

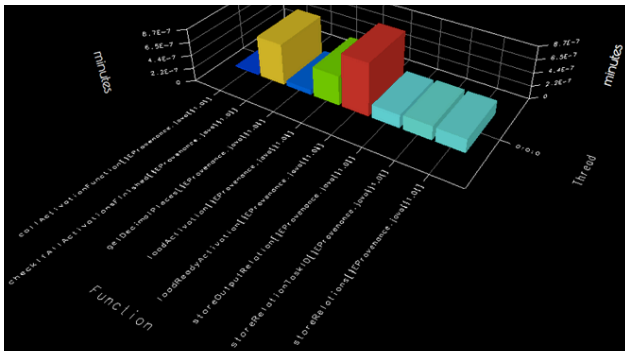

Figure 9. Assessing performance monitoring overhead

5. CONCLUSIONS AND FINAL REMARKS

Performing analytical queries in workflows in distributed environments is an open, yet important, issue. It is fundamental to follow the status of the workflow execution, especially when they execute for weeks or even months. To be aware of the bottlenecks, resource consumption, and other performance issues is mandatory. Most SWfMS already provide some level of monitoring capabilities. However, the type of information provided by these monitoring mechanisms is limited to the amount of activations executed, the volume of data transferred, the average execution time of activities, etc. This type of information is very useful to debug an execution or to fine-tune the environment for a workflow execution. We also provide an important opportunity to understand the behavior of the data derivation based on the performance and resource consumption metrics. When performance data and resource consumption data are not related to domain data, users may not see that a certain data value is presenting an anomalous behavior.

This paper proposes an approach that integrates provenance data, domain data, performance information and resource consumption information in the same integrated database. To achieve this, we propose *PerfMetricEval,* a component for capturing performance and resource consumption data using specialized tools such as SAR and TAU. We integrated *PerfMetricEval* to Chiron, a SWfMS that already collects provenance and domain-specific data in the same database. We evaluated the proposed approach by monitoring the Montage workflow and performing analytical queries that mix different types of data, thus leaving room for domain specialists and code developers to fine tune activities such as investigating input and output data for that particular activity execution when its execution is taking too long. Using the Montage workflow, we also show that the overhead is negligible when compared to the total time needed to execute the workflow. In our use case it was around 0.6% of the total workflow execution. Although the results are promising we still have to evaluate Chiron + *PerfMetricEval* in other scientific experiments in a larger scale. Despite our component has been only integrated to Chiron SWfMS, we intend to adapt/integrate it to other SWfMS in the near future. The only restriction is that the SWfMS should store provenance in a database, like Pegasus and Swift/T do.

6. ACKNOWLEDGMENTS

Authors would like to thank CNPq and FAPERJ for partially funding this work. This research made use of Montage, which is funded by the National Science Foundation (NSF).

7. REFERENCES

[1] T. Oden, R. Moser, and O. Ghattas, "Computer Predictions with Quantified Uncertainty, Part I," in *SIAM News*, 2010, vol. 43(9).

[2] I. J. Taylor, E. Deelman, D. B. Gannon, and M. Shields, *Workflows for e-Science: Scientific Workflows for Grids*, 1st ed. Springer, 2007.

[3] E. Walker and C. Guiang, "Challenges in executing large parameter sweep studies across widely distributed computing environments," in *Workshop on Challenges of large applications in distributed environments*, Monterey, California, USA, 2007, pp. 11–18.

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

[5] E. Deelman, K. Vahi, G. Juve, M. Rynge, S. Callaghan, P. J. Maechling, R. Mayani, W. Chen, R. Ferreira da Silva, M. Livny, and K. Wenger, "Pegasus, a workflow management system for science automation," *FGCS*, vol. 46, pp. 17–35, 2015.

[6] E. Ogasawara, J. Dias, V. Silva, F. Chirigati, D. Oliveira, F. Porto, P. Valduriez, and M. Mattoso, "Chiron: A Parallel Engine for Algebraic Scientific Workflows," *CCPE*, vol. 25, no. 16, pp. 2327–2341, 2013.

[7] H. A. Nguyen, D. Abramson, T. Kipouros, A. Janke, and G. Galloway, "WorkWays: interacting with scientific workflows," *CCPE*, vol. 27, no. 16, pp. 4377–4397, Nov. 2015.

[8] M. Zaharia, M. Chowdhury, M. J. Franklin, S. Shenker, and I. Stoica, "Spark: cluster computing with working sets," in *USENIX conference on Hot topics in cloud computing*, Boston, 2010, pp. 10–17.

[9] M. Mattoso, J. Dias, K. A. C. S. Ocaña, E. Ogasawara, F. Costa, F. Horta, V. Silva, and D. de Oliveira, "Dynamic steering of HPC scientific workflows: A survey," *FGCS*, vol. 46, pp. 100–113, May 2015.

[10] E. Deelman and A. Chervenak, "Data Management Challenges of Data-Intensive Scientific Workflows," in *CCGRID*, 2008, pp. 687–692.

[11] A. Ailamaki, "Managing scientific data: lessons, challenges, and opportunities," in *SIGMOD*, New York, NY, USA, 2011, pp. 1045–1046.

[12] J. Dias, G. Guerra, F. Rochinha, A. L. G. A. Coutinho, P. Valduriez, and M. Mattoso, "Data-centric iteration in dynamic workflows," *FGCS*, vol. 46, pp. 114–126, 2015.

[13] V. Silva, D. de Oliveira, P. Valduriez, and M. Mattoso, "Analyzing related raw data files through dataflows," *CCPE*, vol. 28, no. 8, pp. 2528–2545, 2016.

[14] K. A. C. S. Ocaña, D. Oliveira, E. Ogasawara, A. M. R. Dávila, A. A. B. Lima, and M. Mattoso, "SciPhy: A Cloud-Based Workflow for Phylogenetic Analysis of Drug Targets in Protozoan Genomes," in *Advances in Bioinformatics and Computational Biology (BSB)*, Berlin, Heidelberg, 2011, vol. 6832, pp. 66–70.

[15] B. Balis, M. Bubak, and B. Łabno, "Monitoring of Grid scientific workflows," *Scientific Programming*, vol. 16, no. 2–3, pp. 205–216, 2008.

[16] D. Gunter, E. Deelman, T. Samak, C. H. Brooks, M. Goode, G. Juve, G. Mehta, P. Moraes, F. Silva, M. Swany, and K. Vahi, "Online workflow management and performance analysis with Stampede," in *CNSM*, 2011, pp. 1 –10.

[17] S. S. Shende, "The TAU Parallel Performance System," *International Journal of High Performance Computing Applications*, vol. 20, no. 2, pp. 287–311, May 2006.

[18] K. Savla, T. Temple, and E. Frazzoli, "Human-in-the-loop vehicle routing policies for dynamic environments," in *CDC*, 2008, pp. 1145–1150.

[19] L., Groth, Paul Moreau, *Provenance: an introduction to PROV*. Morgan & Claypool Publishers, 2013.

[20] R. Prodan, S. Ostermann, and K. Plankensteiner, "Performance analysis of grid applications in the ASKALON environment," in *2009 10th IEEE/ACM International Conference on Grid Computing*, 2009, pp. 97–104.

[21] J. S. Vockler, G. Mehta, Y. Zhao, E. Deelman, and M. Wilde, "Kickstarting remote applications," in *International Workshop on Grid Computing Environments*, 2007.

[22] M. Albrecht, P. Donnelly, P. Bui, and D. Thain, "Makeflow: a portable abstraction for data intensive computing on clusters, clouds, and grids," in *Proceedings of the 1st ACM SIGMOD Workshop on Scalable Workflow Execution Engines and Technologies*, 2012, p. 1.

[23] *Monitoring with Ganglia*, 1 edition. Sebastopol, CA: O'Reilly Media, 2012.

[24] A. Jain, S. P. Ong, W. Chen, B. Medasani, X. Qu, M. Kocher, M. Brafman, G. Petretto, G.-M. Rignanese, G. Hautier, D. Gunter, and K. A. Persson, "FireWorks: a dynamic workflow system designed for high-throughput applications," *CCPE*, vol. 27, no. 17, pp. 5037–5059, 2015.

[25] J. C. Jacob, D. S. Katz, G. B. Berriman, J. C. Good, A. C. Laity, E. Deelman, C. Kesselman, G. Singh, M.-H. Su, T. A. Prince, and R. Williams, "Montage: a grid portal and software toolkit for science-grade astronomical image mosaicking," *IJCSE*, vol. 4, no. 2, pp. 73–87, 2009.

[26] E. W. Greisen and M. R. Calabretta, "Representations of world coordinates in FITS," *Astronomy and Astrophysics*, vol. 395, no. 3, pp. 1061–1075, 2002.