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

Vítor Silva[§], Leonardo Neves[§], 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, Rio de Janeiro, 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 scientific programs chained in a coherent flow and executed in High Performance Computing environments. These executions may present 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-grained performance statistics such as the System Activity Report (SAR). However, these tools do not associate their results to their corresponding dataflows. Such analysis is fundamental to trace back the data origins of an error. In this paper, we propose to couple a workflow monitoring data approach to parallel code-profiling tools for workflow executions. The goal is to profile and debug parallel workflow executions by querying a database that is able to integrate performance, resource consumption, provenance, and domain data from simulation programs at runtime. We have implemented our data monitoring approach as a software component that was coupled to TAU and SAR code profiling tools. We show how querying the resulting integrated database enables domain-aware runtime steering of performance anomalies by using the astronomy Montage workflow, as a motivating example. We observe that the overhead introduced by our approach is negligible.

Keywords

Performance analysis; debugging; scientific workflow; provenance.

1. INTRODUCTION

A workflow is an abstraction that defines a set of activities and a dataflow among them [1]. 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 use of High Performance Computing (HPC) or High-Throughput Computing (HTC) environments allied to parallelization techniques such as data parallelism or parameter sweep [2].

To support the modeling and execution of workflows in those environments, standalone parallel Scientific Workflow Management Systems (SWfMS) were developed, such as Swift/T [3], Pegasus [4] and Chiron [5], or SWfMS embedded in Science gateways such as WorkWays [6]. 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 that produces 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 [8] or Spark [7].

It is far from trivial to monitor and steer performance of the resource consumption related to domain data during the parallel execution of workflows [8]. Users need to relate performance and resource consumption information with domain 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 raw 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.

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 [10], which has been coupled to Pegasus, are also able to monitor the execution of workflows in HPC environments at runtime. However, this coarse grain information prevents users 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) [11]. TAU instruments the application code to capture performance data and invokes ParaProf for presenting these data, for instance, using 3D visualizations. Other tools, such as System Activity Report (SAR)[1], 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 raw data files), the user may not see that a certain data value from a huge file is presenting an anomalous behavior.

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

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

relational database management system at runtime. In [12] and [13], 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. Moreover, we also contribute in this paper by developing 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.

This paper is organized in five sections. Section II discusses related work. Section III describes our approach for performance and resource consumption monitoring and the integration between *PerfMetricEval* with Chiron SWfMS. We also show the evaluation of the proposed approach in this section using the Montage workflow in a cluster environment. Section IV concludes the paper and presents some final remarks.

2. RELATED WORK

Related work is organized in two broad categories, systems that monitor the execution of workflows and tools for monitoring low level information on performance and resource consumption. There are several SWfMS that provide monitoring and performance analysis mechanisms within their engines. ASKALON [13], Swift/T, Pegasus (kickstart tool [14]), Makeflow [15] and Chiron provide monitoring mechanisms for users to follow the execution of the workflow and to analyze its behavior. Swift/T and Pegasus provide interfaces to follow the execution while Chiron allows for database queries to be submitted at runtime for workflow 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 the Kickstart tool to do performance analyses that can also be associated to provenance data. Makeflow is a workflow approach that enables performance monitoring and debugging on HPC environments, such as application elapsed time. Chiron provides information about the amount of activations, their execution times, and domain data in the same database, but it does not provide performance metrics neither resource consumption data. STAMPEDE [10] is a SWfMS-independent solution that provides a common model for workflow monitoring, however it does not consider domain-specific data. These SWfMS solutions fail to combine domain data and workflow execution data with performance and resource consumption information.

WorkWays [6] and FireWorks [17] enable users to monitor the status and the elapsed time of tasks, and correlate those data to domain-specific data. They also display performance data related to memory usage and I/O operations. Those performance data allow for identifying performance bottlenecks in HPC environments, however, they are not related to domain data.

There are other approaches that provide detailed performance and resource consumption information for applications (*i.e.* disconnected from the workflow concept). Tuning and Analysis Utilities (TAU) [11] 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. Similar to TAU, DARSHAN is a resource consumption profiler that monitors I/O operations in applications with a non-intrusive solution.

System Activity Report (SAR) is a Linux monitor command that informs system loads, including CPU activity, memory usage statistics, *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 [2] presents a resource monitor tool for gathering performance data during the execution of applications, which enables visualization of performance data, such as the memory usage and % of CPU usage. Ganglia [18] is a distributed monitoring system for distributed infrastructures. Ganglia captures performance information from infrastructure and 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.

Therefore, we observe that existing approaches do provide valuable information for computer science experts to debug a code or to understand the performance of a workflow execution or to follow a scientific workflow execution in an HPC environment. However, when using those existing solutions, users may miss important opportunities to understand the behavior of data derivation based on resource consumption information. When resource consumption data are not related to domain data, it may be hard to find which specific data value is presenting an anomalous memory consumption or execution behavior and act directly on this. In data-intensive workflows this lack becomes really an issue.

3. THE PERFMETRICEVAL COMPONENT

TAU is a tool that supports debugging and profiling of HPC scientific applications for computer experts, like instrumenting the application code to capture performance data and to present these data using a graphical representation. However, users from the application domain also 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 the provenance 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.

In this paper we consider metrics such as total elapsed time (that can be decomposed to identify bottlenecks related to the computational 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. We decomposed the total workflow elapsed time (*T_wf)*, 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 $T\_wf = T\_comp + T\_comm + T\_prov$.

[2] http://ccl.cse.nd.edu/software

For 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*).

PerfMetricEval captures performance and resource consumption metrics using both TAU and SAR. Using TAU, we capture the elapsed time of computing, communication, and provenance operations. However, since TAU does not provide memory, CPU and I/O statistics, we get those from SAR.

The integration of Chiron with *PerfMetricEval* is based on inserting an invocation of *PerfMetricEval* component (after the execution of each activation) to gather fine-grained 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 [11]. *PerfMetricEval* execution flow is presented in Figure 1. In https://github.com/hpcdb/PerfMetricEval, the component is available for download with explanations on how to configure a database and invoke Chiron with *PerfMetricEval.*

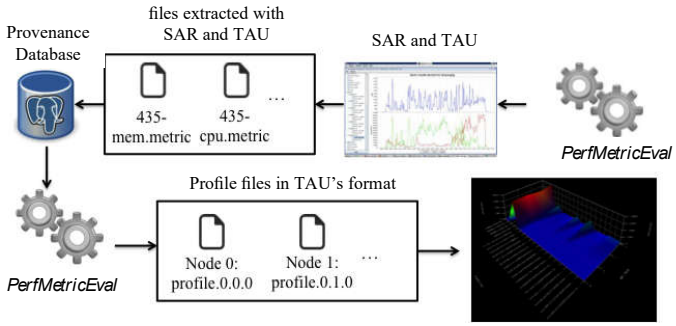

Figure 1. The *PerfMetricEval* execution flow

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, it is created a new file and stored in the workflow workspace. By creating those files, *PerfMetricEval* is able to parse them, extract performance information, and asynchronously load it into the provenance database.

After loading the performance and resource consumption data into the same database, the *PerfMetricEval* component 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 the execution. The TAU visualization tool (paraprof command) graphically displays the generated files. Since the used format is compatible with this 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.

We use the well-known scientific workflow Montage [19], presented in Figure 2, from the astronomy domain as the case study of *PerfMetricEval*. 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. Based on these queries, we register that the experimental results refer to a workflow execution of 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). The Montage execution consumed 1,585 input file images, which produced 17,503 activations that were executed in parallel.

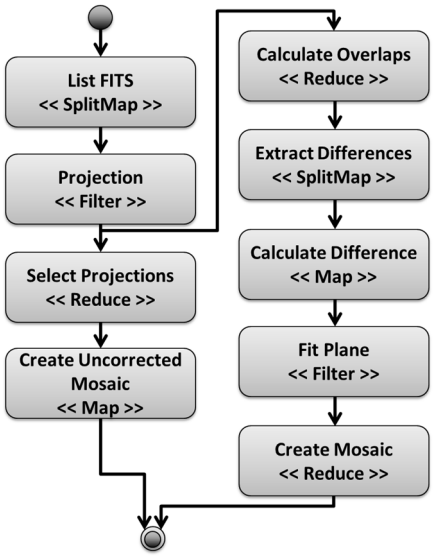

Figure 2. Montage workflow

Analyzing Figure 3 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 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.

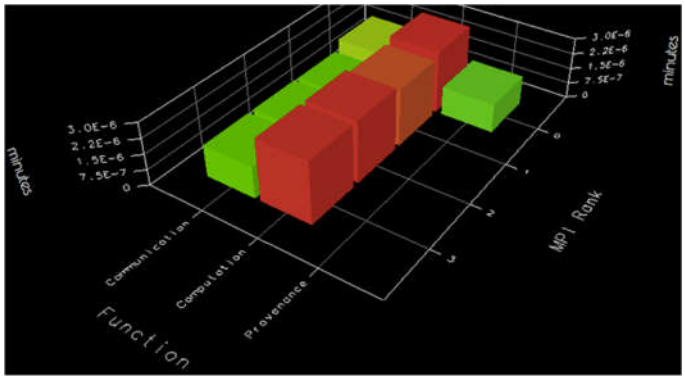

Figure 3. The computing time of each machine

Another important analysis considers activity average resource consumption per machine. Besides, users need to relate the resource

consumption and the domain-specific data. Domain scientists commonly have a fairly good execution time estimate 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 with the corresponding domain files and parameter data of that particular anomaly. For example, it is well known for Montage users that the image region of interest can impact the performance and resource consumption of workflow activations. Thus, users often need to analyze the behavior of a specific subset of the 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 region of interest. In this small scale of Montage workflow execution with Chiron, we observed a generation of 10,647 files. Finding which file has the region of interest with the anomalous behavior is very error prone when the performance data is separated from the workflow execution data and domain data. With the data integration, one query can retrieve the average memory consumption, the average memory used, the average CPU usage, the average CPU usage for one operational system, the average amount of disk blocks read/write, *etc*. All these data are associated with the domain attributes CRPIX1 and CRPIX2 and their corresponding FITS file ids. 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 or on the data file contents. The data extracted from these queries allowed us to generate several TAU graphs. In Figure 4 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 use, considering the metrics *idle* (in light blue), *iowait* (dark blue), *sys* (green), and *user* (red). However, the same behavior is not found when we consider both memory and disk usage statistics.

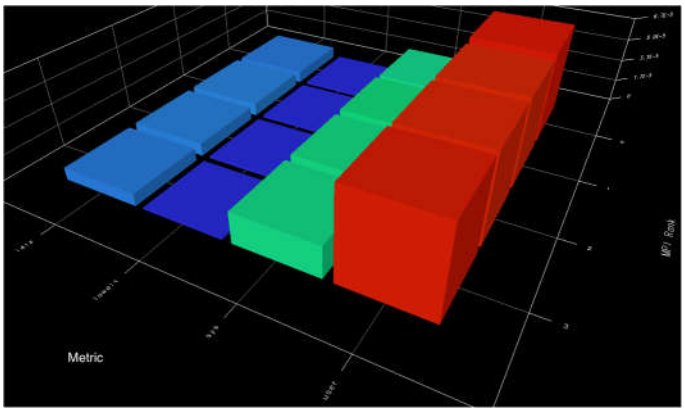

Figure 4. The CPU statistics for activations executed

4. 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 essential. Most SWfMS already provide some level of monitoring capabilities. However, their monitoring mechanisms are limited to following the amount of activations executed, the volume of data transferred, the average execution time of activities, etc. In this paper, we provided 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 introduce *PerfMetricEval,* a component for capturing performance and resource consumption data using specialized tools such as SAR and TAU. We integrated *PerfMetricEval* to Chiron SWMS. We evaluated the present 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 noticed that the overhead is negligible when compared to the total time needed to execute the workflow without *PerfMetricEval*. In this experiment, it was around 0.6% of the total workflow execution. Although the results are promising, we still have to evaluate Chiron + *PerfMetricEval* in large scale scientific experiments. 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 needs to store provenance in a database, like Pegasus and Swift/T do.

5. ACKNOWLEDGMENTS

Authors would like to thank CNPq, FAPERJ, HPC4E (EU H2020 Programme and MCTI/RNP-Brazil, grant no. 689772), and Intel for partially funding this work. This research made use of Montage, which is funded by the National Science Foundation (NSF). Leonardo Neves is currently at Language Technologies Institute, Carnegie Mellon University, Pittsburgh, PA, USA.

6. REFERENCES

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

[2] 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.

[3] 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.

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

[5] 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.

[6] 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.

[7] 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.

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

[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] 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.

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

[12] 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.

[13] 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.

[14] 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.

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

[16] 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.

[17] A. Jain, S. P. Ong, W. Chen, B. Medasani, X. Qu, M. Kocher, M. Brafman, G. Petretto, G.-M. Rignane, 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.

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

[19] 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.