Linking provenance with system logs: a context aware information integration and exploration framework for analyzing workflow execution

Elias el Khaldi Ahanach, Spiros Koulouzis, Zhiming Zhao
elias.el.khaldi@gmail.com, {S.Koulouzis|Z.Zhao}@uva.nl
Informatics Institute,
University of Amsterdam,
Amsterdam, The Netherlands

***Abstract*—When executing scientific workflows in a distributed environment, anomalies of the workflow behavior are often caused by a mixture of different issues, e.g., careless design of the workflow logic, buggy workflow components, unexpected performance bottlenecks or resource failure at the underlying infrastructure. The provenance information only defines data evolution at the workflow level, which does not have an explicit connection with the system logs provided by the underlying infrastructure. Analyzing provenance information and apposite system metrics requires expertise and a considerable amount of manual effort. Moreover, it is often time-consuming to aggregate this information and correlate events occurring at different levels in the infrastructure. In this paper, we propose an architecture to automate the integration among the workflow provenance information with the performance information collected from infrastructure nodes running workflow tasks. Our architecture enables workflow developers or domain scientists to effectively browse workflow execution information together with the system metrics, and analyze contextual information for possible anomalies.**

I. Introduction

In the last decades, researchers have been using sophisticated *research support environments* for efficient data discovery, experiment management, and workflow composition and execution. These research support environments typically include:

1) e-Infrastructures, e.g., EGI[1] and EUDAT[2], focus on the management of the *service lifecycle* of computing, storage and network resources, and provide services to research communities or other user groups to provision dedicated infrastructure and to manage persistent services and their underlying storage, data processing and networking requirements.
2) Research infrastructures (RIs) are facilities, resources, and services constructed for specific scientific communities to conduct research. They can include scientific equipment, knowledge-based resources such as collections, archives or scientific data. Some RIs examples include the Integrated Carbon Observation System (ICOS)[3] for carbon monitoring in atmosphere, ecosystems and marine environments, the European Plate Observing System (EPOS)[4] for solid earth science and Euro-Argo[5] for collecting environmental observations from large-scale deployments of robotic floats in the world's oceans.
3) Virtual Research Environments (VREs) provide user-centric support for discovering and selecting data and software services from different sources, and composing and executing application workflows [1], [2], also referred to as Virtual Laboratories or Science Gateways [3].

Although roles and functions of these different kinds of environments may substantially overlap, we can distinguish that e-infrastructures focus on generic Information and communications technology (ICT) resources (e.g., computing or networking), RIs manage data and services focused on specific scientific domains, and VREs support the lifecycle of specific research activities. Although the boundaries between these environments are not always entirely clear (often sharing services for infrastructure and data management [4]), collectively they represent an important trend in many international research and development projects.

Research support environments combine multiple resources including federated clouds and repositories that allow the sharing of service-oriented architecture (SOA) based scientific workflows [5]. These environments also offer the means to store provenance data concerning the execution of scientific workflows.

When performing an experiment using a Workflow Management System (WFMS) in a VRE, different types of contextual information can be collected:

- Provenance information provided by the workflow system.
- Application logs monitored by the platform (e.g., by Apache Tomcat or Java virtual machine).
- System logs collected by the infrastructure monitoring systems.

[1]http://www.egi.eu/
[2]http://www.eudat.eu/
[3]https://www.icos-ri.eu/
[4]https://www.epos-ip.org/
[5]http://www.euro-argo.eu/

Provenance (PROV)[6] is a typical model often used for workflow provenance. It models causality of workflow events using concepts of agents, entities, and activities involved in data evolution [6]. It has been used in workflow systems like *Apache Taverna* to export information on workflow executions. Provenance can be stored using XML or RDF[7] standards, with many tools available for parsing and querying the data.

At the same time, monitoring systems provide metrics about the usage of e-Infrastructures resources, e.g., CPU usage, memory consumption, and network traffic. Those metrics can be useful for workflow developers to investigate application behavior at the low-level resources, e.g., locating workflow failures caused by the underlying infrastructure resource. However, the provenance and system metrics differ in scope of information and are provided by different sources, which makes the integrated analysis difficult and time-consuming.

In this paper, we propose a context-aware information integration and exploration framework for users to effectively investigate possible workflow execution bottlenecks by combining provenance with the system logs. We discuss an architecture that will allow scientists or service developers to analyze the execution of service-based scientific workflows and visualize possible bottlenecks related with the infrastructure by getting a detailed view of the resource usage of each workflow task. By taking advantage of this information, infrastructure administrators, service developers or scaling controllers may configure the provisioned visualized infrastructure.

The rest of the paper is organized as follows: In section II we discuss the motivation for our solution. Section III presents an overview of related works. Our proposed system architecture is given in Section IV while Sections V and VI are devoted to the assessment of our proposed solution. The paper concludes with Section VII.

II. Motivation

By abstracting the application logic of steps or processes in an experiment, the WFMS allows scientists to efficiently construct, execute and validate complex application at a high level[7].

The adoption of cloud technologies together with the increasing popularity of containerization and DevOps practices have made the development, deployment, and monitoring of web services faster and more efficient.

Scientific workflows usually interact with multiple web services which are often hosted in different cloud infrastructures and may be composed of numerous tasks[8]. When executing such complex workflows, it is often hard to detect the underlying cause of execution bottlenecks which are related to the performance of the infrastructure. If we break down the infrastructure into multiple abstraction levels, we can see that workflows are created and executed on the highest level while resource usage is measured on the lowest levels[9]. As a result, the measured resource metrics are often unreachable and obscured for the user. Although the use of cloud technologies provides scientists with a dedicated infrastructure which combines data and computation [10] along with sophisticated monitoring tools, it is very difficult for a scientist or service developer to discover which Virtual Machine (VM) or container is responsible for execution bottlenecks or failed workflow exertions.

To be able to bridge the gap between the workflow abstraction level and the dedicated infrastructure we set the following objectives :

1) Analyse the execution time of service-based scientific workflows
2) Detect bottlenecks that cause workflow performance degradation
3) Detect the cause of bottlenecks
4) Present the analysis to a user
5) Make resource scaling suggestions to be used by scaling controllers

III. Related Work

In the past, there have been many efforts to detect the sources of performance loss or detect anomalies in the infrastructure while executing scientific workflows.

In [11] the authors propose an approach that aims to help workflow end-users and middleware developers to understand the sources of performance losses when executing scientific workflows in Grid environments. To achieve that they propose a model for estimating the ideal lowest execution time of a workflow and then calculate the total overhead as the difference between the workflow's measured Grid execution time and its ideal time. This work is focused on Grid environments and depends on the presence of a specific scheduler named GRAM [12] to collect the state of each workflow task submitted to a grid site. Moreover, in Grid environments, the performance of the node that is assigned to execute a workflow task is more or less stable. Once a workflow task is scheduled on a node it has exclusive use of its resources.

Also focusing on Grid environments, the authors of [13] presented a simple for autonomous detection and handling of operational incidents in workflow activities. In this work, the authors attempt to classify the state of each task in a workflow and apply the appropriate rule in case of an error. In this work issues like resource, scaling are not addressed since Grid environments assign tasks in a static resource. In [14] the authors have gathered eight months of workflow activity from the e-BioInfra platform and present an analysis on task failures in their e-Infrastructure. Although this work provides some useful insight into the behavior of workflow tasks, it is not connecting the higher level workflow deception tasks with the underlying usage of resources.

The work presented in [15] proposes an online mechanism for detecting anomalies while executing scientific workflows on networked clouds. The authors use an integrated framework to collect online monitoring time-series data from workflow tasks and the infrastructure. This approach is tightly coupled

[6]https://www.w3.org/TR/prov-overview/

[7]https://www.w3.org/RDF/

with the Pegasus WFMS which depends on specific worker nodes to execute workflows tasks.

Existing solutions are outdated and depend on specific task schedulers or WFMS. Although traditionally scientific workflows preserve provenance information, they are never used together with information collected from the infrastructure nodes to analyze contextual information for possible anomalies.

IV. Architecture

Our architecture, named Cross-context Workflow Execution Analyzer (CWEA) enables workflow developers or domain scientists to effectively browse workflow execution information together with the system metrics, and analyze contextual information for possible anomalies. To achieve that it relies on the following components:

- Workflow Context Data Retriever (WCDR): this component queries the provenance data generated by the WFMS to extract the name, start-time, and end-time of each web service call described in the workflow. The WCDR also parses the workflow itself to extract the endpoints of the web services used and the type of call made (e.g., GET, POST, etc.).
- Resource Context Data Retriever (RCDR) this component uses the service endpoints obtained by the WCDR to query the corresponding hosts and retrieve available performance data within the web service's call time-ranges. The performance data typically include CPU utilization, memory and network usage.
- Workflow Execution Analyzer (WFEA): this component abstracts a set of diagnostic algorithms that are used to analyze various performance metrics to find correlations between workflow execution and resource usage. Currently, we have implemented an algorithm that identifies the most time-consuming web services within the context of a workflow execution. The findings of the WFEA are presented in the interactive GUI for visualization or can be consumed in the form of JSON by other software components such as infrastructure scale controllers.
- Interactive GUI: this is a web-based interface that allows the user to combine and visualize the workflow execution steps with the performance metrics of the underlying resources.

Figure 1 shows the overall architecture and its individual components. Each of the components is implemented as a RESTful microservice with its own functionality. For our prototype the WCDR is able to parse t2flow workflows, a specification used by the Taverna WFMS. Also, our microservice architecture design allows us to also support other workflow specifications like SCUFL2 by implementing additional parses.

As mentioned above the WCDR is parsing the provenance data to obtain the execution trace of a workflow. Therefore, complex workflow statures such as loops or conditions are already recorded by the provenance data. As a next step the

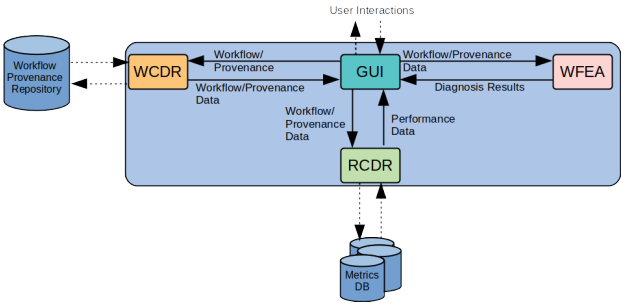


Fig. 1. The CWEA is made by five main components: 1) the WCDR for querying provenance data and parsing workflows, 2) the RCDR for querying performance data, 3) the WFEA for implementing diagnosis algorithms and 4) the Interactive GUI for presenting the results.

WCDR parses the workflow to simply extract web service endpoints and the type of call made.

To be able to query more data sources for performance data, the RCDR may include additional implementations[8]. It is therefore necessary that each VM is hosting a performance metrics collector and a metrics database.

The GUI component is the only component that is accessible from the outside. Besides acting as a graphical interface for the user, the back-end of the GUI component is a REST API for calls made by other applications. This design assures both manual and programmatic interaction.

For our prototype, the provenance data and workflow are manually uploaded by the user to the GUI. However, with our design we aim to be able to connect to provenance data workflow repositories.

To be able to analyze and visualize potential bottlenecks in the execution of a workflow a user should perform the following steps:

1) Once the WFMS, in our case Taverna, has executed a workflow the user exports the workflow's provenance as a file.
2) Once the execution is over the user uploads the provenance and workflow files to the GUI
3) The GUI sends the files to the WCDR where it parses the file and returns for each service in the workflow: 1) its name, 2) its endpoint, 3) its invocation start-time and 4) its invocation end-time. This information is returned in the form of a list and visualized by the GUI. This list can be filtered to select the hosts the user wishes to analyze further.
4) The user specifies the hosts to be analyzed and sends a request to the RCDR via the GUI to gather the relevant performance data. The RCDR attempts to query to databases on the endpoints to retrieve the performance data bound by the timestamps of each web service invocation.
5) Once the performance data are available to WFEA, it performs its analysis and returns the results to the GUI.

[8]At the moment we support the Prometheus database

The sequence diagram in Figure 2 shows the process described above.

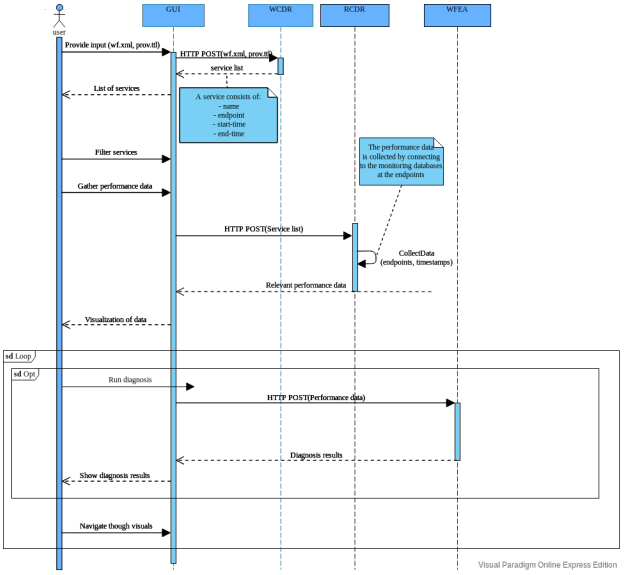


Fig. 2. A sequence diagram showing the interactions between the different components. We assume the use case in which an actor is using the GUI to gather and analyze performance data, giving workflow descriptions as input.

V. Experiments

For our experiments, we hosted several services on three distributed VMs. We used Taverna to create and execute a workflow comprising these services each implementing a set of methods the exhaust the system resources. By doing this, we simulated heavy CPU/MEM/NET use that can be traced back to the performance metrics. To conduct our experiment we used the following components:

- **The Taverna WFMS** where we composed and executed a workflow. We also used Taverna to extract the workflow provenance data.
- **Three VMs** (labeled A, B and C) each containing:
  - **A web service** that offers the following methods: 1) a lightweight call which requires very little resources form the VM, 2) a CPU intensive and 3) a memory intensive [9]. Using these methods, we can simulate heavy CPU and memory usage that can be traced back to the performance metrics.
  - **A performance metrics collector**. For our experimental setup, we used cAdvisor [16], a daemon that collects, aggregates, processes, and exports a range of system metrics about running containers. By default, these metrics include CPU, Memory, and Network.
  - **A metrics database**. In our setup, we used Prometheus[17], a time-series database which is used to gather and store performance metrics from cAdvisor. We use this database to query the metrics concerning the workflow execution.

[9]Both methods use the command stress-ng for 15 sec.

- **The CWEA** and its components to parse the workflow, gather the relevant metrics from different hosts and to perform the visualization.

Since we set up our experiments using VMs, we opted for Docker to run all of these components as separate containers. Deployment of Docker containers is often used in DevOps with the combination of virtualized resources as they offer exclusive access to the resources (e.g., VMs. Moreover, a wide range of tools for monitoring is available as Docker containers. Therefore, it requires minimum effort to set up a realistic performance monitoring framework on each VM. Singularity [18] is another option for containerization of applications which focuses more on high-performance computing (HPC) by providing access to devices like GPUs or MPI hardware. Nevertheless, the wide adoption of Docker together with a wide range of tools and the fact that the scope of this work is beyond HPC made us choose Docker.

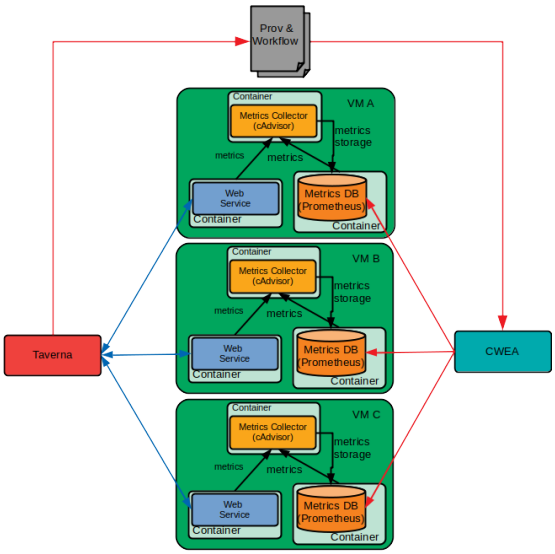


Fig. 3. This diagram depicts our experimental setup. We use three VMs (named A, B and C). On each VM we hosted the a simple web service, cAdvisor and Prometheus. After the workflow execution the user provides to the CWEA (Figure 1 shows its architecture) the provenance and workflow information to query Prometheus on each VM.

Figure 3 shows the configuration of each VM. We used Taverna to create and execute a test workflow shown in Figure 4. We examined published Taverna workflows[10] to create a workflow that has a realistic structure. As a result, we constructed our test workflow comprising a total of six tasks ending in one output port. The workflow contains both sequential and parallel executions. The tasks that are executed are of the following types: 1) a lightweight, 2) a CPU intensive and 3) a memory intensive.

VI. Results

We have executed the workflow described in the previous section and analyzed with CWEA. Figure 5 show the results

[10]Taken from www.myexperiment.org

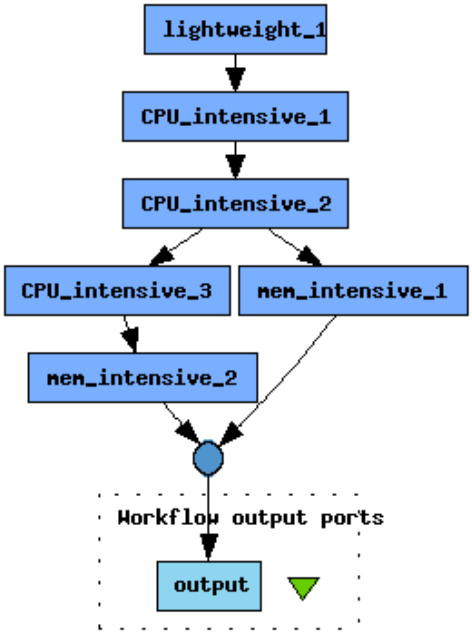


Fig. 4. A simple workflow made of six tasks spread over three VMs. Tasks lightweigt_1, CPU_intensive_2 and mem_intensive_2 where hosted on VM A. Tasks CPU_intensive_1 and mem_intensive_1 on VM B. Task CPU_intensive_3 was hosted on VM C.

as visualized by the GUI. In Figure 5(a) we see the workflow execution analysis. This table provides to the user a view of the workflow execution with a table that shows the name of the task, its endpoint, the HTTP method and the start and end times of each invocation. Next, in 5(b) we see the execution timeline of the workflow and the time required to execute each task. In this timeline, each task is presented by a different color bar which is also highlighted in the resource usage graphs below. The position and length of each bar correspond to the start time and duration of each task. Table I shows the execution times in more detail.

Task Name	Exec. Duration, sec	Perc. of Total Exec., %
mem_intensive_1	26.04	22.34
mem_intensive_2	24.57	21.09
CPU_intensive_1	22.83	19.59
CPU_intensive_3	21.80	18.71
CPU_intensive_2	21.11	18.11
lightweight_1	0.19	0.16

TABLE I
EXECUTION TIMES AND PARENTAGE OF TOTAL EXECUTION FOR EACH TASK IN THE WORKFLOW.

Figure 6 shows the metrics collected from each VM. All sub-figures highlight each service execution (when each service call started and ended) with a separate color which corresponds to the colors shown in Figure 5(b). More specifically, Figure 6(a) presents the CPU used by each task highlighted with the color that corresponds to each task based on the timeline shown in Figure 5(b). In this Figure, the x-axis is the percentage of the CPU used and the y-axis the time of the recorded metric. Similarly in Figure 6(b) we present the memory used by each task, also color-highlighted. In this graph, the x-axis shows the memory used in MB and the y-axis the time. Figures 6(c) and 6(d) show the network usage where the x-axis represent incoming or outgoing data in KB/s and the y-axis time.

Considering the performance of task CPU_intensive_1 we see in Figure 6(a) that it started at approximately 21:45:5 and ended at 21:45:28 and used more than 80% of CPU. However, in Figure 6(b) we see that the same task did not require any memory from its hosting VM, but it did use some network resources as it can be seen from Figure 6(d).

From the results presented here, we can see that the most time-consuming task of the workflow presented in Section V is the mem_intensive. Looking at the results in Figure 6, we see that this task is using CPU, memory and network resources which may attribute for the increased execution time.

VII. Conclusions and Future Work

In this paper, we have presented and evaluated our proposed architecture that allows scientists or service developers to analyze the execution of service-based scientific workflows, visualize possible bottlenecks related with the infrastructure by getting a detailed view of the resource usage of each workflow task. By taking advantage of our solution infrastructure administrators, service developers or scaling controllers my configure the provisioned visualized infrastructure.

As described in Sections IV our proposed architecture is relying on the WFMS to collect provenance data. This means that the workflow execution needs to complete before CWEA can use the provenance data since. However, being able to collect provenance data as each task is completed will greatly benefit our architecture as results would be able to be presented as on the fly. Such an approach requires further investigation on how to use the appropriate APIs from a multiple WFMS or abstract this process by relying on a provenance repository that will be able to collect provenance data as workflow tasks are completed.

Another issue that will require our attention in the future is the persistence of the performance data on each VM. It can be the case that as soon as the workflow execution is over or the VMs are no longer used they may be deleted causing the loss of all performance data. In future cases with the combination of on the fly data gathering as desired above performance data shall be copied to a separate performance database. Having performance data from many workflow executions will also enable us to make use of statistical and AI algorithms to detect and predict possible workflow execution failures due to errors in the resource infrastructure.

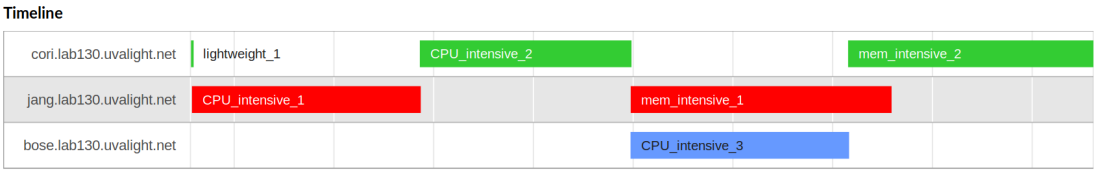
Acknowledgment

This work was supported by the European Union's Horizon 2020 research and innovation programme under grant agreements No. 824068 (ENVRI-FAIR), 654182 (ENVRIPLUS project), 825134(ARTICONF), 676247 (VRE4EIC project),643963 (SWITCH project).

Output

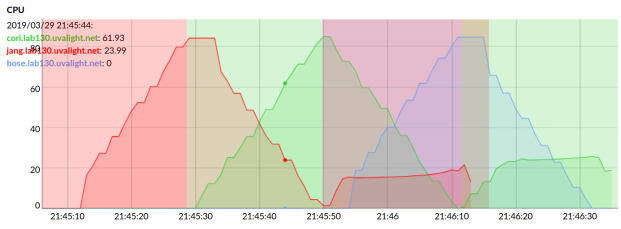
Name	Endpoint	Method	Start-time	End-time
lightweight_1	http://cori.lab130.uvalight.net:8080	GET	2019-2-29 21:45:5.614	2019-2-29 21:45:5.801
CPU_intensive_1	http://jang.lab130.uvalight.net:8080/exhaustCPU	POST	2019-2-29 21:45:5.819	2019-2-29 21:45:28.648
CPU_intensive_2	http://cori.lab130.uvalight.net:8080/exhaustCPU	POST	2019-2-29 21:45:28.666	2019-2-29 21:45:49.775
CPU_intensive_3	http://bose.lab130.uvalight.net:8080/exhaustCPU	POST	2019-2-29 21:45:49.791	2019-2-29 21:46:11.591
mem_intensive_1	http://jang.lab130.uvalight.net:8080/exhaustMEM	POST	2019-2-29 21:45:49.811	2019-2-29 21:46:15.851
mem_intensive_2	http://cori.lab130.uvalight.net:8080/exhaustMEM	POST	2019-2-29 21:46:11.605	2019-2-29 21:46:36.178

(a)

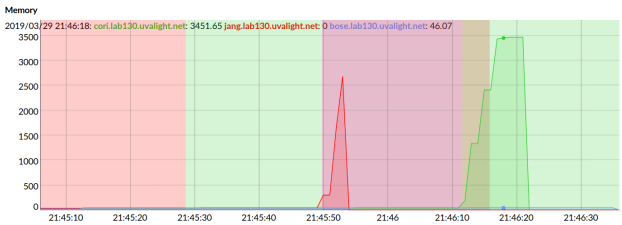


(b)

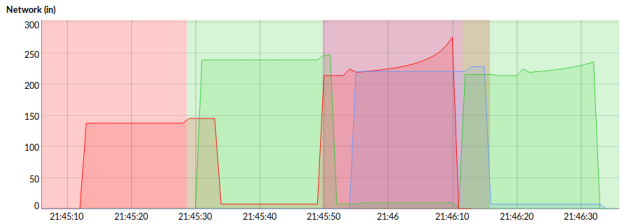
Fig. 5. Workflow execution analysis and execution timeline as presented to the user by the GUI. The execution timeline provides to the user an overview of the time taken to execute each task.



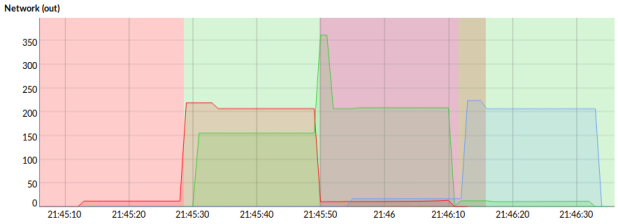
(a) CPU resources: This graph shows the CPU resources utilized by each service during the workflow execution.



(b) Memory resources: This graph shows the memory resources utilized by each service during the workflow execution.



(c) Incoming network traffic: This graph shows the incoming data for each task, execution.



(d) Outgoing network traffic: This graph shows the outgoing data for each task, execution.

Fig. 6. Combined results after parsing the workflow, querying the provenance file and querying the relevant resource usage. All sub-figures highlight each service execution with a separate color which corresponds to the colors shown in Figure 5(b)

REFERENCES

[1] L. Candela, D. Castelli, and P. Pagano, "Virtual research environments: an overview and a research agenda," *Data Science Journal*, vol. 12, no. 0, pp. GRDI75–GRDI81, 2013.

[2] Z. Zhao, A. Belloum, C. De Laat, P. Adriaans, and B. Hertzberger, "Distributed execution of aggregated multi domain workflows using an agent framework," *Services, 2007 IEEE Congress on*, pp. 183–190, 2007.

[3] M. A. Miller, W. Pfeiffer, and T. Schwartz, "The cipres science gateway: enabling high-impact science for phylogenetics researchers with limited resources," *Proceedings of the 1st Conference of the Extreme Science and Engineering Discovery Environment: Bridging from the eXtreme to the campus and beyond*, p. 39, 2012.

[4] S. Koulouzis, A. S. Belloum, M. T. Bubak, Z. Zhao, M. ivkovi, and C. T. de Laat, "Sdn-aware federation of distributed data," *Future Generation Computer Systems*, vol. 56, pp. 64 – 76, 2016.

[5] K. Evans, A. Jones, A. Preece, F. Quevedo, D. Rogers, I. Spasić, I. Taylor, V. Stankovski, S. Taherizadeh, J. Trnkoczy, G. Suciu, V. Suciu, P. Martin, J. Wang, and Z. Zhao, "Dynamically reconfigurable workflows for time-critical applications," *Proceedings of the 10th Workshop on Workflows in Support of Large-Scale Science*, pp. 7:1–7:10, 2015.

[6] P. Groth and L. Moreau, "Prov-overview. an overview of the prov family of documents," 2013.

[7] R. Cushing, S. Koulouzis, A. Belloum, and M. Bubak, "Applying workflow as a service paradigm to application farming," *Concurrency and Computation: Practice and Experience*, vol. 26, no. 6, pp. 1297–1312, 2014.

[8] R. F. da Silva, R. Filgueira, I. Pietri, M. Jiang, R. Sakellariou, and E. Deelman, "A characterization of workflow management systems for extreme-scale applications," *Future Generation Computer Systems*, vol. 75, pp. 228–238, 2017.

[9] Y. Demchenko, Z. Zhao, P. Grosso, A. Wibisono, and C. De Laat, "Addressing big data challenges for scientific data infrastructure," in *4th IEEE International Conference on Cloud Computing Technology and Science Proceedings*, pp. 614–617, IEEE, 2012.

[10] S. Koulouzis, D. Vasyunin, R. Cushing, A. Belloum, and M. Bubak, "Cloud data federation for scientific applications," in *Euro-Par 2013: Parallel Processing Workshops* (D. an Mey, M. Alexander, P. Bientinesi, M. Cannataro, C. Clauss, A. Costan, G. Kecskemeti, C. Morin, L. Ricci, J. Sahuquillo, M. Schulz, V. Scarano, S. L. Scott, and J. Weidendorfer, eds.), (Berlin, Heidelberg), pp. 13–22, Springer Berlin Heidelberg, 2014.

[11] R. Prodan and T. Fahringer, "Overhead analysis of scientific workflows in grid environments," *IEEE Transactions on Parallel and Distributed Systems*, vol. 19, pp. 378–393, March 2008.

[12] I. Foster, "Globus toolkit version 4: Software for service-oriented systems," *Journal of computer science and technology*, vol. 21, no. 4, p. 513, 2006.

[13] R. Ferreira da Silva, T. Glatard, and F. Desprez, "Self-healing of operational workflow incidents on distributed computing infrastructures," in *2012 12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (ccgrid 2012)*, pp. 318–325, May 2012.

[14] S. Madougou, S. Shahand, M. Santcroos, B. van Schaik, A. Benabdelkader, A. van Kampen, and S. Olabarriaga, "Characterizing workflow-based activity on a production e-infrastructure using provenance data," *Future Generation Computer Systems*, vol. 29, no. 8, pp. 1931 – 1942, 2013. Including Special sections: Advanced Cloud Monitoring Systems & The fourth IEEE International Conference on e-Science 2011 e-Science Applications and Tools & Cluster, Grid, and Cloud Computing.

[15] P. Gaikwad, A. Mandal, P. Ruth, G. Juve, D. Krl, and E. Deelman, "Anomaly detection for scientific workflow applications on networked clouds," in *2016 International Conference on High Performance Computing Simulation (HPCS)*, pp. 645–652, July 2016.

[16] "cadvisor (container advisor), official github page." https://github.com/google/cadvisor. Accessed: 2019-03-28.

[17] "Prometheus, an open-source systems monitoring and alerting toolkit.." https://prometheus.io/docs/introduction/overview/. Accessed: 2019-03-28.

[18] G. M. Kurtzer, "Singularity 2.1. 2-linux application and environment containers for science, 2016," *Available from Internet:¡ https://doi. org/10.5281/zenodo*, vol. 60736.