Functionality Testing in the Automation of Scientific Application Workflows in an HPC Environment

Felipe de Jesús Orozco Luna, Jesús Manuel Alemán González, and Veronica Lizette Robles Dueñas(✉)

University of Guadalajara (CADS Data Analysis and Supercomputing Center), Zapopan, Jalisco, México
{forozco,lizette}@cads.udg.mx, felipe.orozco@academicos.udg.mx, jesus.aleman4182@alumnos.udg.mx

Abstract. This paper presents the results obtained in performance tests of task automation in the supercomputing cluster of the University of Guadalajara (CADS Data Analysis and Supercomputing Center).

The main objective was to design an automated workflow process to take advantage of high computational performance in scientific applications, routines in R, Python or shell that by nature consume only one core, and that by the volume of data to be processed could allow the execution of multiple tasks at once in a supercomputing cluster environment, or even in the cloud, for an efficient use of the infrastructure. In addition to using Singularity containers to encapsulate applications or scripts to be used in workflows.

The following tools were tested: Snakemake as a tool for workflow automation and scaling, as well as Singularity container technologies for application encapsulation and SLURM for managing resource usage in the cluster.

The results are presented as well as the experience gained in using these technologies.

Keywords: Automation · Snakemake · Singularity containers · Scaling · Reproducibility · Supercomputing cluster · Parallelization

1 Introduction

We are currently faced with the need to process large amounts of data with tools that use a single processing core, which makes scalability and automation difficult.

This problem is very common in the scientific context, as many scientific applications use a single processing core for a large amount of data. For this reason, and because of the computational resources required, these applications are candidates for use in a processing flow that enables scalability and automation.

To address this issue, tests were conducted to quantify the performance and scaling of the parallelized tools. These tests were performed by creating a container in Singularity with a scientific application.

We test the functionality and performance of the tools required for workflow automation, scaling, and reproducibility. The following themes motivate our study:

P. Navaux et al. (Eds.): CARLA 2022, CCIS 1660, pp. 220–232, 2022.
https://doi.org/10.1007/978-3-031-23821-5_16

- Framing within an automated processing flow with containers. Within containers, scientific applications that require only a processing core or routines in R, Python, or shell.
- Create a suitable environment for executing tasks, many of which cannot inherently be parallelized but require large amounts of data and information processing and storage capacity.
- Create processing workflows that are scalable and automated.
- Design and test workflows with parallelized tasks for optimal use of resources provisioned in the supercomputing cluster.
- Use Python scripts integrated as processing boxes (one core per job) in these processing flows.

The present study helped us to test and design processing flows with different scenarios of tools and cluster capacities, testing since one compute node using partially all available cores (up to 36 cores per node), and performance tests using a maximum of 28 compute nodes.

Parameters to be tested included the scalability and automation capabilities of a workflow to make the best use of the supercomputing cluster resources.

An additional motivation is to gain experience using these processing schemes for CADS and supercomputing users who wish to leverage the use of processing in these work scenarios that inherently lack scalability and parallelization capabilities.

2 Infrastructure Used

The Data Analysis and Supercomputing Center (CADS) is a space created by the University of Guadalajara, located in the facilities of the University Center for Economic and Administrative Sciences, in Zapopan Jalisco, México.

Within the CADS is the University of Guadalajara Supercomputer, associated with large processing, storage and communications capabilities, whose purpose is to enable and accelerate scientific research and technological development of the university community.

Up to 28 compute nodes of the Leo ATROX Supercomputing equipment were used for these tests, each with the following characteristics:

2 Xeon-6154 (SKYLAKE) processors with 18 cores at 3.0 Ghz Between 188 - 392 GB RAM

3 Tools Used

3.1 Slurm

Slurm is an open source tool used in supercomputing environments that handles the management and allocation of resources within the cluster. It is based on a configuration file where you can specify the amount of resources to use along with the routines or commands for processing your work.

Slurm's operation is based on queue management, where each job is assigned an identifier and given a priority for execution. Slurm is in charge of managing the job contention in the cluster, since it has the ability to send jobs in parallel. Due to this operation, Slurm allows to start, stop and monitor the jobs in a very simple way, so that the user knows at all times in which state his job is.

3.2 Singularity

Singularity is an open source platform for the management, creation and execution of containers within local and supercomputing environments. It was created primarily for the encapsulation of computationally demanding projects, therefore, it has become an ideal tool to be installed in an HPC environment.

Compared to other container platforms, Singularity offers greater security, scalability and reproducibility, in addition, it allows transforming Docker images to its format to be used and executed with Singularity. Last but not least, Singularity can communicate perfectly with other tools that allow you to improve the development of your project and optimize aspects of it.

3.3 Snakemake

Snakemake is a workflow management system for creating scalable and reproducible data analysis. It is based on Python because it is so simple and easy for humans to understand.

Snakemake creates a workflow based on rules that contain the input files to be processed to create the final output files, as well as the scripts or commands needed to process the workflow.

Snakemake also lets you adjust the amount of computing resources needed to run the workflow. This allocation depends heavily on the project and the amount of resources the user has on their machine. However, Snakemake is excellent for supercomputing environments, for running projects with high computational requirements.

Automation, reproducibility and scalability are the words that sum up Snakemake. Finally, Snakemake adapts to many tools. For example, you can use Singularity, Kubernetes and Slurm in your workflow, making the extension and combination of technologies even more powerful.

Summary of tools used and their versions. See Table 1.

4 Design of the Processing Flow for Testing

Several processing flows were designed for these performance tests:

- A Singularity container was used by taking a Docker image with a scientific tool used for high-energy particle simulation.
- In addition to Python routines for NumPy data processing.

Additional considerations are:

Table 1. Versioning used for testing.

Tool	Version	Source
SLURM	20.11.3	https://slurm.schedmd.com/
Singularity	3.6	https://docs.sylabs.io/guides/3.6/user-guide/
Snakemake	6.15.5	https://snakemake.readthedocs.io/en/v6.15.5/getting_started/installation.html
Centos OS	8.2	https://www.centos.org/

- A processing box is defined as a routine application, script, or library that can execute inside or outside a container, generally receiving input, performing processing, and returning output.
- The tests consist of two stages of processing. The first stage uses a processing box in a Singularity container with the capacity to execute a large number of tasks simultaneously and without dependencies on each other; and the next stage uses Python processing boxes that depend on the output data from the previous stage to perform the tests, see detail in Fig. 1.
- In both processing boxes mentioned above, only one computational core was consumed per task.
- The tests were performed with 360, 720 and 1,000 files as output data for data processing.
- The maximum number of compute nodes used for the performance test was 28.

5 Analysis of Possible Cases

5.1 Running Python Script

The benefits of using tools like Snakemake to automate and scale work processes in a supercomputing cluster with containers like Singularity can be seen in Table 2, which shows the different tool scenarios.

It is important to note that the grid above the diagonal was not included to avoid repeating the scenarios.

The above leads me to answer the premise with which the tests began: How can I fully exploit the capabilities of a supercomputing cluster in a container environment and/or Python routines? The answer is as follows: Use Snakemake as a tool that supports reproducibility and automation of tasks that result in less time spent running a large number of jobs and using the installed infrastructure.

5.2 Running Python Script with SLURM and Singularity

In the case of using tools such as Snakemake, whether or not using tools such as SLURM or Singularity, we can see the possible processing scenarios. See Table 3.

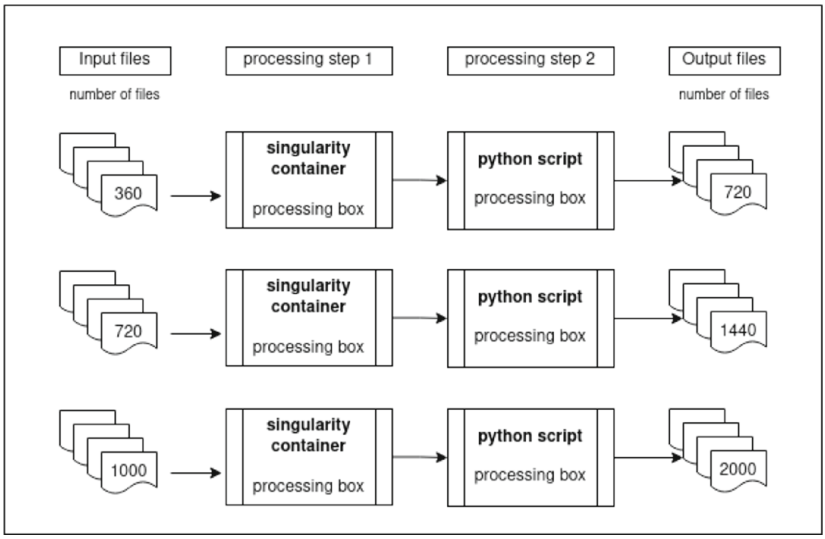

Fig. 1. Details of the designs used in the tests. The number of input files for each test, The number of processing steps executed, At each processing step, a routine may be encapsulated in python, r, shell, or a scientific application that uses only one processing core. The output files, the number of files expected as a result of the processing.

Table 2. Possible capabilities per scenario using Python routine.

Capabilities possible when running Python scripts vs. Tools				
Python Script	**Without SLURM**	**SLURM**	**Snakemake**	**Singularity**
Without SLURM	A single core	-	-	-
SLURM	NA	A single core	-	-
Snakemake	Multiple nodes	Multiple nodes	Multiple nodes	-
Singularity	A single core	A single core	Multiple nodes	A single core

5.3 Running a Singularity Container with Snakemake Using SLURM

This table analyzes the case of using a scientific tool for data processing, running on a single processing core within a Singularity container, and how to make its use more efficient with massive processing and data in the supercomputing cluster. See Table 4.

5.4 Notes for Tables 2, 3 and 4

Single core: Only uses one processing core.

Table 3. Possible capabilities using Python routines inside a container.

Possible Python scripting capabilities inside a container		
Snakemake	**Without SLURM**	**SLURM**
Singularity	A single node	Multiple nodes

Table 4. Capacities using singularity containers.

Send to run APP in container		
Snakemake	**Without SLURM**	**SLURM**
Singularity	A single node	Multiple nodes

Single node: It can only run on one node, using from 1 to 36 cores per node.
Multiple nodes: Ability to run on multiple compute nodes, using multiple cores.

6 Results of Executions

6.1 Case 1:

The following data were used for the first test:

- 360 initial input files, using Snakemake.
- Maximum of 10 compute nodes.
- Using two stages of processing using a Singularity container and Python script to process data in NumPy.

The results of the first test can be seen in Fig. 2, where 360 files were processed. The test started with 36 cores and the processing time was 2,222 s; when the 360 cores were used, the time dropped to 322 s.

Figure 2 shows that while the applications and scripts used only consume one processor core. However, with tools to automate the processing flow such as Snakemake and

a reasonable amount of data, a large number of tasks can be run simultaneously, making the processing time more efficient. It is worth noting that thanks to the functionalities of Snakemake and SLURM, the use of multiple computing nodes is achieved.

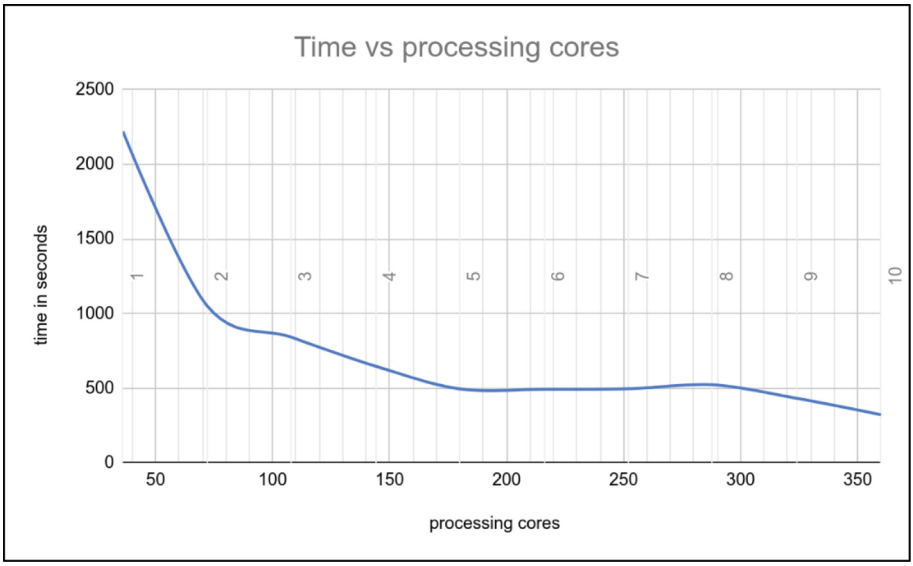

Fig. 2. Processing 360 files. This figure shows that applications scale better from node 1 to node 4, the more cores, the less processing time. From node 5 to node 8, there is no improvement in performance. And from node 9, performance improves again. This behavior is related to the number of processing steps and the routines that are processed in each step.

It is important to note that we want to demonstrate the benefit and advantage of automating applications that do not scale natively and how they can take advantage of the supercomputing cluster infrastructure (Table 5).

6.2 Case 2:

The following data were used for the second test:

- 720 initial input files, using Snakemake.
- Maximum of 20 compute nodes.
- Using two stages of processing using a Singularity container and Python scripts to process data in NumPy.

The results obtained in the second test can be seen in Fig. 3, where 720 files were processed, the test started with 36 cores and the processing time was 5156 s, when using the 720 cores, the time dropped to 447 s.

Table 5. Detail of processing times of 360 files.

Processing time for 360 files	
Number of cores	**Time in seconds**
36	2222
72	1053
108	842
144	648
180	495
216	493
252	496
288	523
324	432
360	322

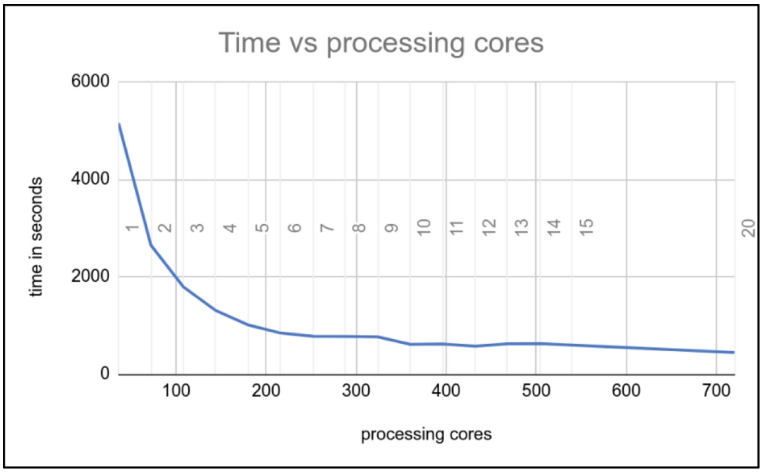

Fig. 3. Processing of 720 files.

6.3 Case 3:

The following data were used for the third test:

- 1000 initial input files, using Snakemake.
- Maximum of 28 computational nodes.
- Use of two processing stages with a Singularity container and Python scripts to process the data in NumPy.

The results of the second test can be seen in Fig. 4. When processing 1,000 files, the test was started with 36 cores and the processing time was 6,567 s; when using the 1,008 cores (28 compute nodes), the time dropped to 567 s.

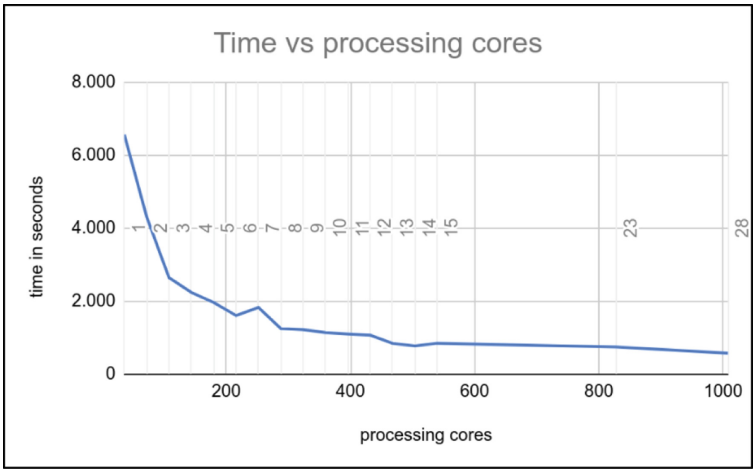

Fig. 4. Processing of 1000 files.

When comparing performance, varying the number of files to process in Fig. 5 shows similar trends of lowering processing time by increasing the number of processing cores.

A comparison of the efficiency of the three runs with different numbers of files is made and compared to an ideal efficiency based on processing per compute node. See Fig. 6.

In this figure, an interesting result can be seen: When testing with 720 files, using 5 to 6 nodes produced an efficiency that is slightly higher than the ideal efficiency.

It should be noted that at the time of submitting jobs to the Slurm queue to be executed, these jobs depend on the handling and management of Slurm on the processing cores, as well as the high demand and utilization of the cluster by other users. This therefore affects the performance of the workflow tasks and causes performance spikes and dips, as can be seen in Fig. 6.

It is assumed that efficiency can be affected by the amount of RAM in each node, although this depends on the applications running in the workflow.

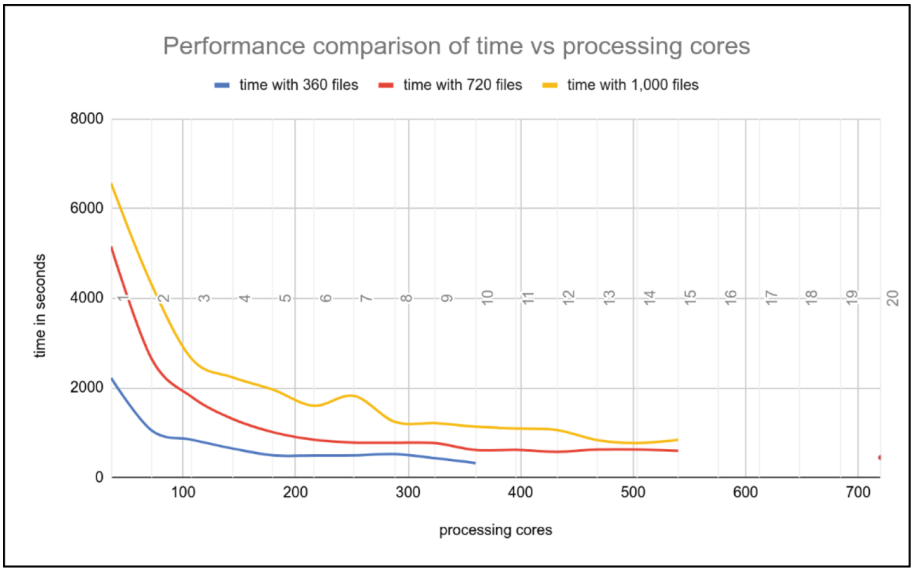

Fig. 5. Performance comparison Scale the processing flow not the application.

7 Discussion of Results and Conclusions

There are important considerations in discussing the results, as these tests and their results compare the improvements in execution times in a framework of parallelizing and automation of a particular use of processing using processing boxes. The first using a tool that consumes only one processing core within a singularity container, and the second processing step using a Python routine that consumes a single processing core. Reviewing the benefits that can be achieved in an automation and parallelization framework in a supercomputing cluster.

Based on the above considerations, it is not intended to serve as a basis for subsequent performance comparisons, but rather to demonstrate the benefits of sharing these applications.

These performance tests should serve as a reference for those who need to build robust automation and parallelization frameworks, even if their tools do not inherently allow parallelization.

Based on the tests carried out with the combination of the 3 technologies used, the following results and observations were obtained:

7.1 Testing Time

For the analysis, the times of Snakemake were used, since it is the tool that manages the entire workflow, from the input files to the generation of the output files, and also the processing times of Slurm. It is worth noting that Slurm manages its own processing times. For this reason, it may happen that Slurm completes a series of files and releases the used resources, but the job queue still appears as a busy node because Snakemake is trying to complete the workflow for that series.

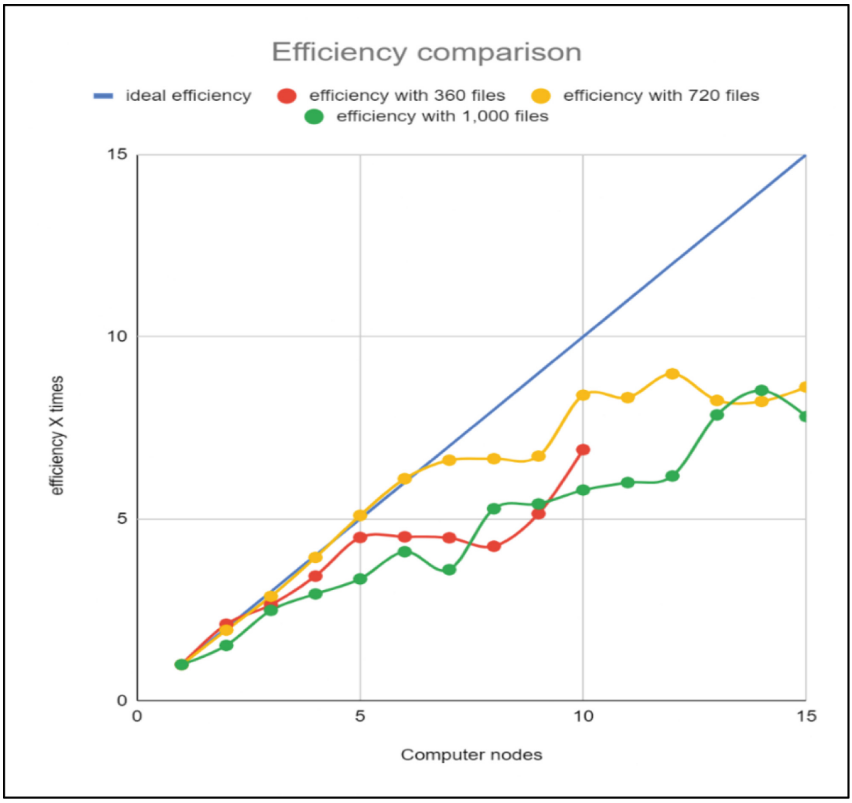

Fig. 6. The ideal efficiency shows the same ratio, in the increase of compute nodes with efficiency gains.

7.2 Duration of Tests

From the design of the processing sequences, the construction of the test container, the design of the automation scripts, as well as the execution of the scripts, up to the analysis of the results, about 3 months were needed.

7.3 What Limitations There Were

Although it is possible to integrate several scientific applications that only consume a processing core, or use R, Python or shell scripts, it will not always be beneficial to integrate them into a processing flow, since there will be the limitation of the volume of data and how they are processed.

One of the observations that had the most impact on the tests and the results of the different scenarios to be run was the management and the amount of cluster resources available, since in some cases resources (compute nodes) have to be shared with other users because the resources that should be used are not provided due to the high demand of users using the cluster.

This resource sharing limitation can occur at any time and is something that we as users cannot control, since Slurm is responsible for resource allocation. For this reason, it is ideal to run the operations in a custom working partition that contains the set N amount of resources to be used, for example not in a general partition where many users use the same resources.

7.4 Learning

In cases where better execution efficiency has been shown, the number of nodes is chosen to match the number of processing cores with the number of jobs to be processed simultaneously, i.e., if 360 files are sent for execution, it would be ideal to take 10 nodes since the infrastructure has 36 cores per node to execute 360 processing cores simultaneously.

This results in greater optimization of the total job duration. Using a large number of nodes is not always the best solution, as this depends on the SLURM queues and the amount of processing in the cluster.

Better performances were shown using the total number of cores per node.

7.5 Conclusions and Benefits

Scaling capabilities were tested with Slurm, Snakemake, and Singularity technologies and found to have parallelization capabilities that can be applied to projects that require large processing capacities and data volumes and are limited by their applications that do not have parallelization capabilities.

Glossary

- NumPy: Python language library used for the creation of vectors and matrices along with the collection of mathematical functions.
- Python: High-level programming language for developing applications of all kinds.
- Container: Technology used for the encapsulation of projects (work environments).
- Docker: Open-source platform for managing and creating containers.
- Kubernetes: Open-source platform for container and microservices management.

References

1. Mölder, F., et al.: Sustainable data analysis with Snakemake. F1000Res. **10**, 33 (2021)
2. How to Manage Workflow with Resource Constraint on HPC. https://www.sichong.site/workflow/2021/11/08/how-to-manage-workflow-with-resource-constraint.html
3. GitHub smk-simple-slurm, smk-simple-slurm. https://github.com/jdblischak/smk-simple-slurm
4. Slurm Workload Manager. https://slurm.schedmd.com/documentation.html
5. Snakemake. https://snakemake.readthedocs.io/en/v6.15.5/
6. Singularity. https://docs.sylabs.io/guides/3.6/user-guide/

7. Sokolov, S., Idiriz, O., Vukadinoff, M., Vlaev, S.: Scaling and automation in cloud deployments of enterprise applications. J. Eng. Sci. Technol. Rev. Special Issue on Telecommunications, Informatics, Energy and Management (2019)
8. Vaquero, L.M., Rodero-Merino, L., Buyya, R.: Association for Computing Machinery. SIGCOMM Comput. Commun. Rev (2011)
9. Caragnano, G., et al.: Scalability of a Parallel Application in Hybrid Cloud. IEEE Computer Society (2014)
10. Sarkar, S., Abdulla, P.P., Ramaswamy, S.: Analysis, evaluation, and assessment for containerizing an industry automation software. In: 2020 IEEE International Conference on Systems, Man, and Cybernetics (SMC), pp. 1972–1979 2020. https://doi.org/10.1109/SMC42975.2020.9282840