SPIRIT: A Microservice-Based Framework for Interactive Cloud Infrastructure Planning

Koulouzis, S.; Bianchi, R.; van der Linde, R.; Wang, Y.; Zhao, Z.

DOI
10.1007/978-3-031-06156-1_32

Publication date
2022

Document Version
Author accepted manuscript

Published in
Euro-Par 2021: Parallel Processing Workshops

License
CC BY

Link to publication

Citation for published version (APA):
Koulouzis, S., Bianchi, R., van der Linde, R., Wang, Y., & Zhao, Z. (2022). SPIRIT: A Microservice-Based Framework for Interactive Cloud Infrastructure Planning. In R. Chaves, D. B. Heras, A. Ilic, & D. Unat (Eds.), *Euro-Par 2021: Parallel Processing Workshops: Euro-Par 2021 International Workshops, Lisbon, Portugal, August 30-31, 2021 : revised selected papers* (pp. 405-416). (Lecture Notes in Computer Science; Vol. 13098). Springer. https://doi.org/10.1007/978-3-031-06156-1_32

SPIRIT: A microservice-based framework for interactive Cloud infrastructure planning

Spiros Koulouzis$^{1,2[0000-0001-8652-315X]}$, Riccardo Bianchi^{1,2}, Robin van der Linde^{1}, Yuandou Wang^{1}, and Zhiming Zhao$^{1,2[0000-0002-6717-9418]}$

[1]Multiscale Networked Systems research group,
University of Amsterdam, the Netherlands
[2]LifeWatch ERIC, Virtual Lab Innovation Center, Amsterdam, the Netherlands
`z.zhao@uva.nl`

Abstract. The IaaS model provides elastic infrastructure that enables the migration of legacy applications to cloud environments. Many cloud computing vendors such as Amazon Web Services, Microsoft Azure, and Google Cloud Platform offer a pay-per-use policy that allows for a sustainable reduction in costs compared to on-premise hosting, as well as enable users to choose various geographically distributed data centers. Using state-of-the-art planning algorithms can help application owners to estimate the size and characteristics of the underlying cloud inveterate. However, it's not always clear which is the optimal solution especially in multi-cloud environments with complex application requirements and QoS constraints. In this paper, we propose an open framework named SPIRIT, which allows a user to include cloud infrastructure planning algorithms and to evaluate and compare their solutions. SPIRIT achieves this by allowing users to interactively study infrastructure planning algorithms by adjusting parameters via a graphical user interface, which visualizes the results of these algorithms. In the current prototype, we have included from the IaaS Partial Critical Path algorithm. By taking advantage of SPIRIT's microservice-based architecture and its generic interfaces a user can add to the framework, new planning algorithms. SPIRIT can transform an abstract workflow described using the CWL to a concrete infrastructure described using the TOSCA specification. This way the infrastructure descriptions can be ranked on various key performance indicators.

Keywords: virtual infrastructure planning · IaC · microservice · IaaS · workflow · IC-PCP

1 Introduction

The cloud computing paradigm allows for on-demand IT service delivery via the internet. The main benefits of cloud computing are dynamic scalability and elasticity, which is achieved via virtualized resources [1].

The IaaS model enables enterprises to migrate their in-house software stack to remote cloud data centers. The IaaS model also provides virtual infrastructures for applications with specific performance requirements [2]. However, selecting Cloud infrastructure services and configuring them for specific objectives is time-consuming for an enterprise. It can also be costly when an enterprise consumes more resources than its application requires.

Most cloud providers offer a pay-per-use policy, which allows for a substantial reduction in costs compared to on-premise hosting. However, cloud providers do not offer tools for estimating and comparing the cost of an application as a function of its requirements, QoS, QoE, etc., and the size of the infrastructure. For instance, Azure[1], Google[2], and Oracle[3] offer similar pricing calculators, that allow for the estimation of costs given a selection of cloud services. The cost of each service is displayed as well as the total costs. However, these tools are limited to just calculating the total costs and cost per service. They only work for their corresponding providers and therefore are not cloud-agnostic. Additionally, they do not help decide which services are needed for an application to preserve performance or other QoS demands. Moreover, the choice of services and their characteristics have to be done manually, since there is not an automated infrastructure recommendation. Moreover, some services may be only available in specific data centers.

The process of designing and formally describing a customized Cloud infrastructure for an application with specific requirements can be described as Virtual Infrastructure planning [3–5]. Virtual infrastructure planning is also often referred to as Infrastructure as a Service planning or even simply *infrastructure planning*. To elaborate on this definition, application developers want to select cloud resources while optimizing certain QoS, such as performance and costs. Furthermore, they want to do this is in a time-efficient way. Within infrastructure planning, resource utilization is an important objective. This is usually achieved by: resource sharing, minimum resource allocation, and load balancing [6].

Infrastructure planning is important in various contexts, such as the cloud computing context, and scientific computing context. There are many approaches to tackle this problem that depends on the context. The most obvious way is manual configuration. This usually involves running the application in the cloud, and keeping adding resources until the performance of the application meets the requirements specified. This method is not efficient, as it requires several time-consuming manual iterations. Also, for more complex applications, such as a complex workflow with many tasks, this approach becomes even less viable [7]. If infrastructure planning is used to migrate an existing on-premise application to the cloud, one can pick services in the cloud with similar specifications as the on-premise services. This is often referred to as resource mapping [4]. Nevertheless, this approach is only viable if there is already an on-premise solution available.

[1] https://azure.microsoft.com/en-us/pricing/calculator/

[2] https://cloud.google.com/products/calculator/

[3] https://www.oracle.com/cloud/cost-estimator.html

[4] https://cloud.google.com/solutions/resource-mappings-from-on-premises-hardware-to-gcp

For other types of applications, such as (scientific) workflows, it is often the case that a list of interdependent tasks needs to be executed. These tasks usually process large data sets and require a specific amount of servers with certain properties [8]. This problem requires a different IaaS planning approach, such as estimating required resources through the use of scheduling algorithms, such as the partial critical path algorithm [4]. We will be looking at existing solutions more extensively in this research.

Aside from the planning itself, it is can also be difficult to effectively compare two IaaS solutions and make decisions suitable for a specific application and its context [9]. This is because there may be differences in performance and QoS in each solution, which can be hard to evaluate [10].

To tackle the above issues, we propose an open framework with the following requirements: 1. the proposed open-framework should allow various infrastructure planning algorithms to be used and analyzed simultaneously, 2. it should be user-centered with an intuitive user interface, 3. it should adhere to DevOps principles by outputting IaC, 4. and it should implement a model to rank the generated infrastructure plans.

Analyzing the above requirements we offer an implementation with the following key contributions: 1. A framework that via its intuitive user interface, allows users to interactively study infrastructure planning algorithms. 2. A framework that can generate multiple infrastructure descriptions for time-critical applications. 3. A framework that is extendable by allowing developers to add their planning algorithms. This allows for a wide range of application types to be planned for. 4. A framework that allows the user to dynamically compare planning results.

The remainder of this paper is organized as follows. Section 2 reviews state-of-the-art infrastructure planning and workflow scheduling algorithms. Section 3 presents the requirements and architecture design of our proposed tool, namely SPIRIT. In section 4, we focus on our usability study results . Finally, Section 5 presents our conclusions and a description of future work.

2 State-of-the-art

The infrastructure planning problem can be approached via the use of scheduling algorithms. Workflow scheduling tries to solve the problem of mapping each task in a workflow to a suitable resource and to order the tasks on each resource to satisfy some performance criterion [11]. However, in this paper, we will use workflow scheduling only to generate infrastructure solutions instead of mapping tasks to virtual machines at run-time.

Abishami et al. [11] introduce the IC-PCP and the IC-PCPD2 algorithms. Where the critical path, is the longest of all execution paths from the beginning to the end in a task graph. [12]. Both algorithms are workflow scheduling algorithms for the cloud.

Taal et al. [4] proposed different implementations, greedy and more stringent, of the IC-PCP algorithm designed by [11]. The paper focuses on getting the cheapest cloud infrastructure while adhering to the deadline of the workflow.

Wang et al. [13] proposed a machine learning-based approach called deep-Q-network to schedule multi-workflows in the cloud. To improve the completion time and user's cost of this approach, a Markov game model is applied, which has the number of workflow applications and VM's as state input and the maximum completion time and cost as rewards. The proposed model is tested via scientific workflows and Amazon EC2, as well as several other algorithms such as non-dominated sorting genetic algorithm-II, multi-objective particle swarm optimization, and game-theoretic-based greedy algorithms.

Rimal et al. [14] proposed a workflow scheduling algorithm for multi-tenant cloud computing environments. The algorithm focuses on minimizing the makespan of workflows, tardiness, cost of execution of workflows, and make use of idle cloud resources.

Wu et al. [15] proposed a task scheduling algorithm based on QoS-driven for cloud computing. The algorithm works by creating a sorted list of tasks based on task attributes, including user privilege, expectation, task length, and the pending time of task in the queue. Then the algorithm will traverse the list and assign the tasks to the services that will complete them the fastest.

Jain et al. [16] compare four static workflow scheduling algorithms, FCSS, Round-Robin, Min-Min, and Max-Min. These algorithms are compared based on a set of parameters, such as makespan, and costs such as communication cost and computation cost. The algorithms are compared by the use of workflows generated by the Pegasus workflow generator and cloud resources.

Visheratin et al. [17] introduce a new, improved algorithm for workflow scheduling called CDCGA. They compared their algorithm to the IC-PCP [11] and the LDD-LS algorithms by running it on the same workflows.

Workflow scheduling algorithms generally require performance models to do their calculations. A performance model contains the execution time, for each task in the workflow, on one of the available machines. So to get useful calculations, we need to make sure that this model is accurate. These performance models are quite ambiguous in the sense that it is hard to predict these for tasks without actually running them. There are existing benchmarking solutions that can be used to obtain performance values [18]. The user of these algorithms might want to know how much of an impact a change in this parameter has on the overall costs and makespan. This is something we will further analyze during the design of our framework.

3 SPIRIT: A microservice-based framework for interactive Cloud infrastructure planning

In this section, we describe requirements and system architecture.

3.1 Requirements

From a developer's perspective, we identified which features to be included in the open framework by identifying user stories. To be specific, the tool should offer one or more IaaS solutions for planning applications. Since we are building an open framework that should provide value to application developers, we need to identify what features should be included from their point of view. This can be achieved by identifying user stories. As a developer, I want: – one or more IaaS solutions for my application, to I save time and costs. – to apply QoS constraints on the provided solution, so my application will behave as intended when it is run on the generated infrastructure. – to generate IaaS solutions that are cloud agnostic, so I can deploy it on various cloud providers. – several IaC solutions, so I have the option to run my application on a different infrastructure. – to add my planning algorithm to the tool, so if I have a better planning approach for my application. – IaaS solutions, in the form of an IaC template,so it can be automatically deployed. – to compare various price and performance models so I can evaluate the impact of these parameters on the result(s). – to customize the virtual machines used by the planning algorithms. – to compare the results on one or more key performance indicators so I can select the solution that is best suited. – to automatically generate planning parameters so I can use the planner without having this data.

Functional Requirements Here we list the functional requirements that we want our framework to fulfill. These requirements describe the desired behavior of our system. We did a MoSCoW analysis on these functional requirements, such that the requirements have an assigned priority. Therefore the system: 1. must integrate planning approaches for at least one type of application, 2. must have a web interface that is accessible via most common browsers, 3. must use common standards for APIs, such as REST, 4. must apply at least two different planning approaches, 5. must provide an IaaS solution in the form of IaC if the QoS demands can be satisfied, 6. must allow the user to configure virtual machines to be considered by the planner, 7. must provide a ranking scheme, in which a recommendation can be made based on KPIs, 8. should allow users to specify the preferred cloud provider, 9. could allow users to insert their planning approach, 10. should allow the user to generate planning parameters based on empirical values, 11. could automatically select the correct planning algorithm based on user input, 12. could visualize the recommended infrastructures.

Non-functional Requirements Besides the functional requirements we also designed some non-functional requirements for our system. These requirements focus on the technical aspects of our system. 1. The graphical user interface should focus on accessibility and usability. 2. The web interface should load within 10s, given an internet speed of >10 mb/s 3. The system should be able to handle at least 1000 simultaneous requests. 4. The system should provide solutions at least 70% of the time if the parameters are within acceptable boundaries.

5. The processing time of the systems should not exceed 5s. 6. The software will be open source and adhere to the software design principles that are included in SOLID.

3.2 Architecture

To fulfill the proposed requirements, we designed an architecture of our system, as can be viewed in Figure 1 and is composed by the elements listed below:

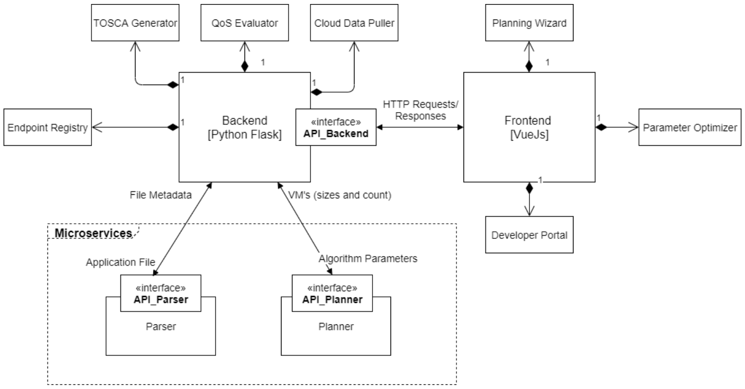


Fig. 1: Architecture Diagram

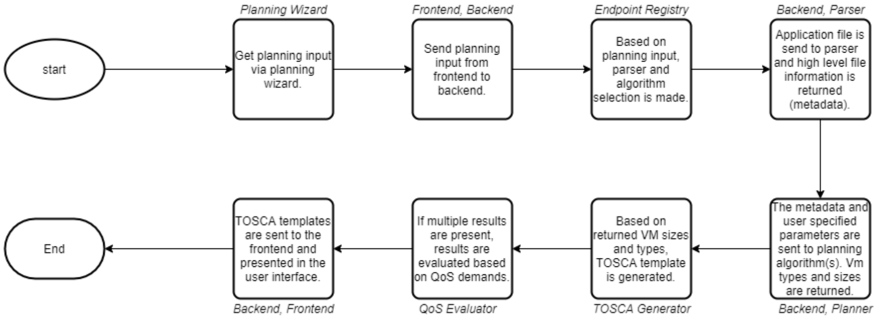
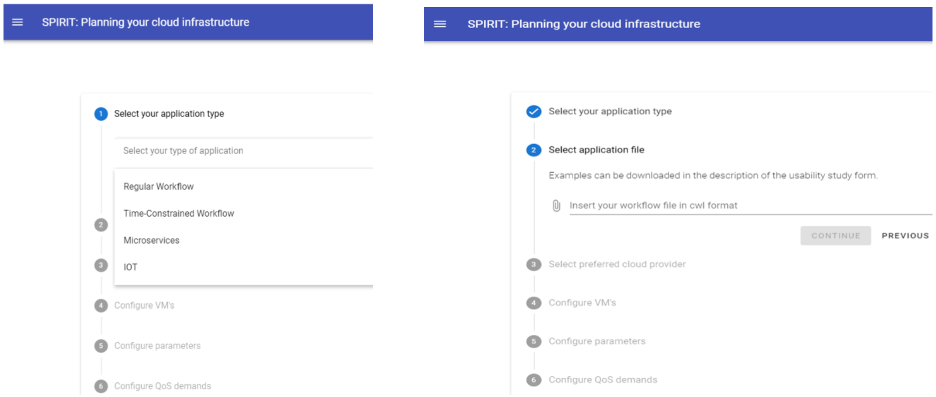

start
Planning Wizard
Get planning input via planning wizard.
Frontend, Backend
Send planning input from frontend to backend.
Endpoint Registry
Based on planning input, parser and algorithm selection is made.
Backend, Parser
Application file is send to parser and high level file information is returned (metadata).
The metadata and user specified parameters are sent to planning algorithm(s). Vm types and sizes are returned.
Backend, Planner
Based on returned VM sizes and types, TOSCA template is generated.
TOSCA Generator
If multiple results are present, results are evaluated based on QoS demands.
QoS Evaluator
TOSCA templates are sent to the frontend and presented in the user interface.
Backend, Frontend
End

Fig. 2: Process Flow and Architectural components

Backend: It provides a RESTful API to the Frontend and allows for the registration of infrastructure planning algorithms

Frontend: This is the web user interface that allows the user to select the type of application they want to plan for (workflow application, time-constrained workflow application, microservice-based application, IOT based application),



SPIRIT: Planning your cloud infrastructure

1 Select your application type

Select your type of application

- Regular Workflow
- Time-Constrained Workflow
- Microservices
- IOT

2

3

4 Configure VM's

5 Configure parameters

6 Configure QoS demands

(a) Select type of application.

SPIRIT: Planning your cloud infrastructure

✓ Select your application type

2 Select application file

Examples can be downloaded in the description of the usability study form.

Insert your workflow file in cwl format

CONTINUE PREVIOUS

3 Select preferred cloud provider

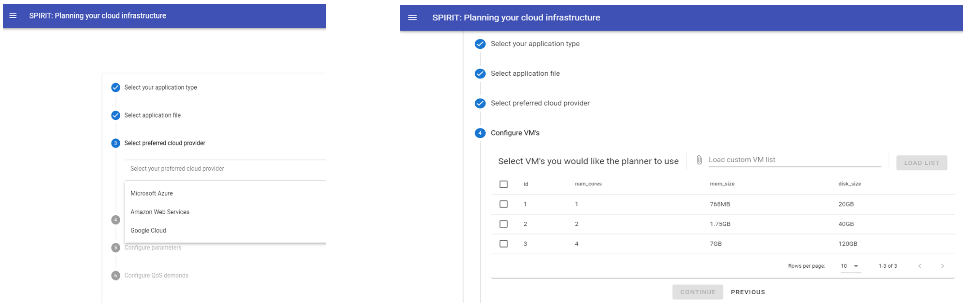
4 Configure VM's

5 Configure parameters

6 Configure QoS demands

(b) Upload application description file.

Fig. 3: SPIRIT wizard for selecting application type and description.



SPIRIT: Planning your cloud infrastructure

✓ Select your application type

✓ Select application file

3 Select preferred cloud provider

Select your preferred cloud provider

- Microsoft Azure
- Amazon Web Services
- Google Cloud

4

5 Configure parameters

6 Configure QoS demands

(a) Select the cloud provider.

SPIRIT: Planning your cloud infrastructure

✓ Select your application type

✓ Select application file

✓ Select preferred cloud provider

4 Configure VM's

Select VM's you would like the planner to use

Load custom VM list

LOAD LIST

☐	id	num_cores	mem_size	disk_size
☐	1	1	768MB	20GB
☐	2	2	1.75GB	40GB
☐	3	4	7GB	120GB

Rows per page: 10 1-3 of 3

CONTINUE PREVIOUS

(b) Select types of VMs.

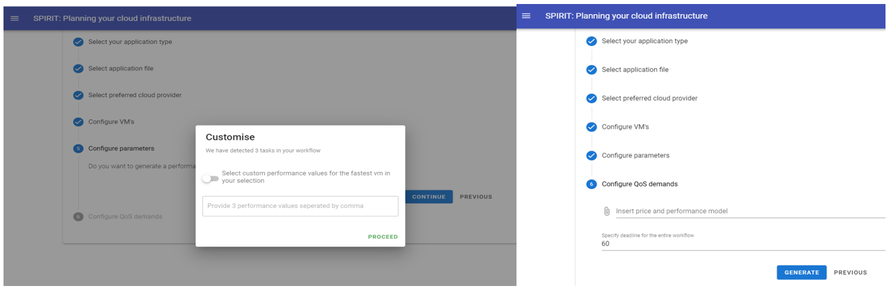
Fig. 4: SPIRIT wizard for selecting cloud provider and VMs.

specify their performance model, and present to the user the results of the planning algorithm. Developers can register additional planning services via the developer portal.

Parser: This component is responsible for analyzing the application description and extracting the parameters need for the planning algorithm. In our proof of concept, we implemented a parser for the CWL.

Planner: This component generates the infrastructure plans. In our proof of concept, we implemented two versions of the IC-PCP algorithm: A greedy version and a greedy version with a repair cycle, as proposed by Zhao et al. [4].

Cloud Data puller: This component retrieves available virtual machines and corresponding prices from several cloud providers. Data are displayed in the user interface, from which the user can make a selection. The selection will be used by the planning algorithms.



SPIRIT: Planning your cloud infrastructure

- Select your application type
- Select application file
- Select preferred cloud provider
- Configure VM's
- 5 Configure parameters

  Do you want to generate a performa

- 6 Configure QoS demands

Customise

We have detected 3 tasks in your workflow

Select custom performance values for the fastest vm in your selection

Provide 3 performance values seperated by comma

SPIRIT: Planning your cloud infrastructure

- Select your application type
- Select application file
- Select preferred cloud provider
- Configure VM's
- Configure parameters
- 6 Configure QoS demands

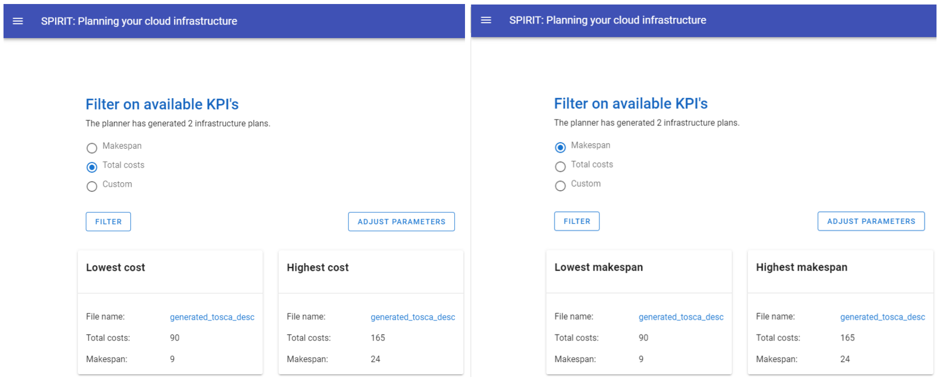
Insert price and performance model

Specify deadline for the entire workflow
60

(a) Select (optional) configuration parameters.

(b) Select (optional) the relevant parameters for the planning algorithm.

Fig. 5: SPIRIT wizard for selecting parameters for the planning algorithm.



SPIRIT: Planning your cloud infrastructure

Filter on available KPI's

The planner has generated 2 infrastructure plans.

○ Makespan
● Total costs
○ Custom

Lowest cost

File name:	generated_tosca_desc
Total costs:	90
Makespan:	9

Highest cost

File name:	generated_tosca_desc
Total costs:	165
Makespan:	24

SPIRIT: Planning your cloud infrastructure

Filter on available KPI's

The planner has generated 2 infrastructure plans.

● Makespan
○ Total costs
○ Custom

Lowest makespan

File name:	generated_tosca_desc
Total costs:	90
Makespan:	9

Highest makespan

File name:	generated_tosca_desc
Total costs:	165
Makespan:	24

(a) Compare the solutions

(b) Compare the solutions

Fig. 6: The SPIRIT infrastructure plan results

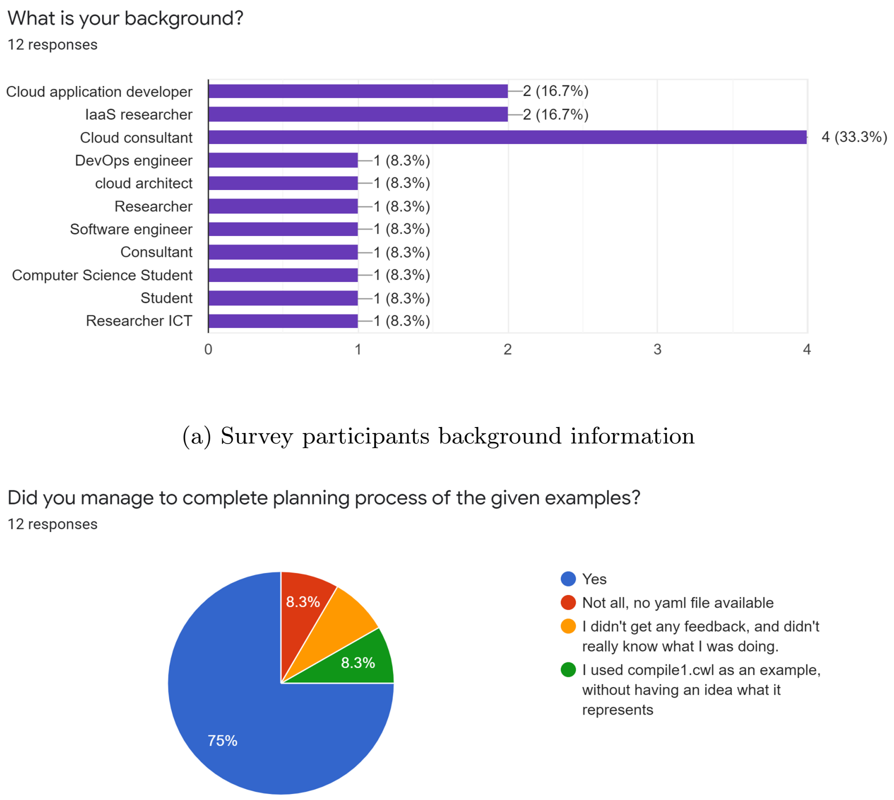
QoS Evaluator: It has the responsibility to process QoS demands from the user. The component will find infrastructure solutions that comply with the specified QoS demands. If there is no solution available that complies, the user will be notified accordingly.

TOSCA Generator: This component is responsible for generating a TOSCA description based on the output from the planning algorithms. This information can be used by a provisioner to automatically set up the infrastructure.

Endpoint Registry: It is used to store the available (external) services.

To illustrate how the components are used and interact with each other, we describe a process flow which is shown in Figure 2, for planning a cloud infrastructure for an application.

According to the flow diagram in Figure 2 a user takes the flowing steps to generate an infrastructure plan: 1. Select type of application you want to plan for (Figure 3a) 2. Upload the application description file. Currently that is a CWL file (Figure 3b). 3. Select the cloud provider (Figure 4a). 4. Select type of



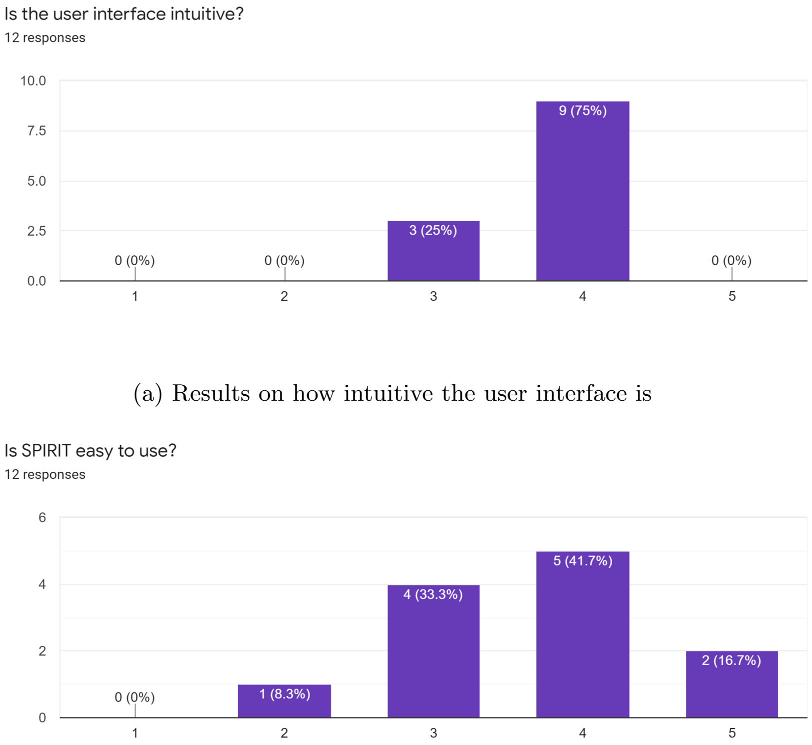
(a) Survey participants background information

(b) Results on completed planing task

VMs for in the infrastructure plan (Figure 4b). 5. Select (optional) configuration parameters to generate a planning input model. This model contains the costs and the performance characteristics of the VMs (Figure 5a). 6. Select (optional) the parameters for the planning algorithms. In the current version, this includes the price and performance model (Figure 5b). 7. Compare the solutions, based on the selected KPIs. In our proof of concept, the user can compare the available solutions on makespan and costs (Figure 6a and 6b).

4 Usability Study

To evaluate the usability of our GUI we surveyed Cloud and DevOps experts. By following the presented guidelines, the user was expected to complete a planning process. The planning process is considered complete when the user has downloaded at least one of the proposed infrastructure descriptions. We also expected the participants to use our ranking scheme to rank the presented output on the



(a) Results on how intuitive the user interface is

(b) Results on how easy to use the user interface is

Fig. 8: Survey results.

available key performance indicators. After completion of the planning process, the user was asked to answer the questions in the survey.

A total of twelve people participated in the experiment. Although the overall volume of participants is low, nearly all the participants have a background relevant to work. As can be seen in Figure 7a, most participants work in the field of Cloud consultancy, or they are Cloud application developers or even IaaS researchers.

Figure 7b indicates that the majority of users were able to complete the planning process except one. According to that user, they did not have enough information to complete the task.

Figure 8a shows how intuitive the user interface is considered by the participants. A score of 1 means it is not intuitive, and a score of 5 means it is very intuitive.

Figure 8b presents the results considering the ease of use of SPIRIT. A linear scale is applied, where a score of 1 means it is hard to use, and a score of 5 very easy to use. The majority is in the range from 3 to 5, which can be considered positive.

5 Conclusion and Future Work

In this paper, we elaborated and implemented an open framework with a user-friendly GUI that can create cost-effective infrastructure plans for various application types. Our proof of concept can transform an abstract workflow defined in CWL to a concrete infrastructure defined using the TOSCA standard. Our microservice-based architecture and generic interfaces make our framework extendable for other planning algorithms, and thus other application types. Our tool also allows for the ranking of the generated infrastructures on various key performance indicators, therefore, allowing for essayer migration of in-house applications to the Cloud while reducing operating costs. In this paper, we have implemented two versions of the IC-PCP algorithm: A greedy version and a greedy version with a repair cycle. In the future, we aim to use SPIRIT as a platform to integrate and compare more planning algorithms.

Acknowledgment

This work has been partially funded by the European Union's Horizon 2020 research and innovation programme by the project CLARIFY under the Marie Sklodowska-Curie grant agreement No 860627, by the ARTICONF project grant agreement No 825134, by the ENVRI-FAIR project grant agreement No 824068, by the BLUECLOUD project grant agreement No 862409, by the LifeWatch ERIC.

References

1. M. Carroll, A. Van Der Merwe, and P. Kotze, "Secure cloud computing: Benefits, risks and controls," in *2011 Information Security for South Africa.* IEEE, 2011, pp. 1–9.
2. Y. Hu, H. Zhou, C. de Laat, and Z. Zhao, "Concurrent container scheduling on heterogeneous clusters with multi-resource constraints," *Future Generation Computer Systems*, vol. 102, pp. 562–573, Jan. 2020.
3. M. P. Anastasopoulos, A. Tzanakaki, and K. Georgakilas, "Virtual infrastructure planning in elastic cloud deploying optical networking," in *2011 IEEE Third International Conference on Cloud Computing Technology and Science.* IEEE, 2011, pp. 685–689.
4. A. Taal, J. Wang, C. de Laat, and Z. Zhao, "Profiling the scheduling decisions for handling critical paths in deadline-constrained cloud workflows," *Future Generation Computer Systems*, vol. 100, pp. 237–249, 2019.

5. Y. Hu, J. Wang, H. Zhou, P. Martin, A. Taal, C. de Laat, and Z. Zhao, "Deadline-Aware Deployment for Time Critical Applications in Clouds," in *Euro-Par 2017: Parallel Processing*, F. F. Rivera, T. F. Pena, and J. C. Cabaleiro, Eds. Cham: Springer International Publishing, 2017, vol. 10417, pp. 345–357, series Title: Lecture Notes in Computer Science.
6. K. N. Georgakilas, A. Tzanakaki, M. Anastasopoulos, and J. M. Pedersen, "Converged optical network and data center virtual infrastructure planning," *Journal of Optical Communications and Networking*, vol. 4, no. 9, pp. 681–691, 2012.
7. Z. Zhao, P. Grosso, J. van der Ham, R. Koning, and C. de Laat, "An agent based network resource planner for workflow applications," *Multiagent and Grid Systems*, vol. 7, no. 6, pp. 187–202, Dec. 2011.
8. K. Vahi, M. Rynge, G. Juve, R. Mayani, and E. Deelman, "Rethinking data management for big data scientific workflows," in *2013 IEEE International Conference on Big Data*, 2013, pp. 27–35.
9. S. Koulouzis, P. Martin, H. Zhou, Y. Hu, J. Wang, T. Carval, B. Grenier, J. Heikkinen, C. de Laat, and Z. Zhao, "Time-critical data management in clouds: Challenges and a dynamic real-time infrastructure planner (drip) solution," *Concurrency and Computation: Practice and Experience*, p. e5269, 2019.
10. J. Frisch, "Comparison of iaas solutions: how companies find the right solution," https://docs.microsoft.com/en-us/learn/modules/predict-costs-and-optimize-spending/2-estimate-costs-with-the-azure-pricing-calculator, 2017, [Online; accessed 15-may-2020].
11. S. Abrishami, M. Naghibzadeh, and D. H. Epema, "Deadline-constrained workflow scheduling algorithms for infrastructure as a service clouds," *Future Generation Computer Systems*, vol. 29, no. 1, pp. 158–169, 2013.
12. M. Rahman, S. Venugopal, and R. Buyya, "A dynamic critical path algorithm for scheduling scientific workflow applications on global grids," in *Third IEEE International Conference on e-Science and Grid Computing (e-Science 2007)*. IEEE, 2007, pp. 35–42.
13. Y. Wang, H. Liu, W. Zheng, Y. Xia, Y. Li, P. Chen, K. Guo, and H. Xie, "Multi-objective workflow scheduling with deep-q-network-based multi-agent reinforcement learning," *IEEE Access*, vol. 7, pp. 39 974–39 982, 2019.
14. B. P. Rimal and M. Maier, "Workflow scheduling in multi-tenant cloud computing environments," *IEEE Transactions on Parallel and Distributed Systems*, vol. 28, no. 1, pp. 290–304, 2016.
15. X. Wu, M. Deng, R. Zhang, B. Zeng, and S. Zhou, "A task scheduling algorithm based on qos-driven in cloud computing," *Procedia Computer Science*, vol. 17, pp. 1162–1169, 2013.
16. A. Jain and R. Kumari, "A review on comparison of workflow scheduling algorithms with scientific workflows," in *Proceedings of International Conference on Communication and Networks*. Springer, 2017, pp. 613–622.
17. A. A. Visheratin, M. Melnik, and D. Nasonov, "Workflow scheduling algorithms for hard-deadline constrained cloud environments," *Procedia Computer Science*, vol. 80, pp. 2098–2106, 2016.
18. O. Elzinga, S. Koulouzis, A. Taal, J. Wang, Y. Hu, H. Zhou, P. Martin, C. de Laat, and Z. Zhao, "Automatic collector for dynamic cloud performance information," in *2017 International Conference on Networking, Architecture, and Storage (NAS)*. IEEE, 2017, pp. 1–6.