SIM-PIPE DryRunner: An approach for testing container-based big data pipelines and generating simulation data

Aleena Thomas
SINTEF AS
Oslo, Norway
Aleena.Thomas@sintef.no

Nikolay Nikolov
SINTEF AS
Oslo, Norway
Nikolay.Nikolov@sintef.no

Antoine Pultier
SINTEF AS
Oslo, Norway
Antoine.Pultier@sintef.no

Dumitru Roman
SINTEF AS
Oslo, Norway
Dumitru.Roman@sintef.no

Brian Elvesæter
SINTEF AS
Oslo, Norway
Brian.Elvesæter@sintef.no

Ahmet Soylu
Oslo Metropolitan University
Oslo, Norway
Ahmet.Soylu@oslomet.no

***Abstract*—Big data pipelines are becoming increasingly vital in a wide range of data intensive application domains such as digital healthcare, telecommunication, and manufacturing for efficiently processing data. Data pipelines in such domains are complex and dynamic and involve a number of data processing steps that are deployed on heterogeneous computing resources under the realm of the Edge-Cloud paradigm. The processes of testing and simulating big data pipelines on heterogeneous resources need to be able to accurately represent this complexity. However, since big data processing is heavily resource-intensive, it makes testing and simulation based on historical execution data impractical. In this paper, we introduce the SIM-PIPE DryRunner approach – a dry run approach that deploys a big data pipeline step by step in an isolated environment and executes it with sample data; this approach could be used for testing big data pipelines and realising practical simulations using existing simulators.**

***Index Terms*—Big data pipelines; Dry run; Software containers; Sandbox; Testing; Simulation**

I. Introduction

The need for supporting big data pipeline processing is increasing rapidly with more and more applications running on the Cloud and large IoT systems handling huge volumes of data [1]. Big data pipelines are designed to handle large amounts of streaming and batch processing data and are becoming indispensable in a wide variety of application domains [2]. One of the main challenges in managing big data pipelines is analyzing the behaviour of different pipeline steps in order to deploy them in a cost-effective manner. Since deploying computing resources for these pipelines is expensive, it is crucial to adjust the deployment parameters for optimized execution and to ensure only required resources are provisioned [3]. Therefore, one of the key aspects of the big data pipeline lifecycle relates to testing and simulation before deployment in a production setting [4]. Testing refers to executing steps in a pipeline according to its definition, whereas simulation focuses on estimating the performance of the pipeline in the actual computing infrastructure by predicting the performance of the pipeline given the execution parameters. An efficient mean of testing and simulating pipelines before deployment allows identifying errors and bottlenecks early and addressing them before provisioning expensive computing resources in the actual production environment on the Cloud-Edge continuum.

There are multiple simulation solutions for big data pipelines (e.g., [5]–[7]). One of the main challenges with the simulators is that most of the existing approaches rely on results from previous runs of pipelines or analyses by an expert in order to make predictions [4]. In the case of big data, predicting performance using previous runs is likely to result in high costs if the pipeline is highly computing-intensive. Big data pipelines are complex and dynamic processes built to run on top of a multitude of heterogeneous services and computing resources, which makes prediction of their performance a challenge [2]. To this end, we propose an approach—SIM-PIPE DryRunner—based on *dry running* of big data pipelines. We describe *dry running* of big data pipelines as the execution of a pipeline using a sample or smaller input data size (compared to the full-scale big data) on a test environment as opposed to using the infrastructure for production deployment.

The overall approach is depicted in Figure 1. We assume that the resource usage metrics for the dry run of the pipeline on a representative set of small input data can be used in the analysis of its behaviour for large amounts of input data. The proposed approach deploys each step in the correct order in an isolated testing environment, hereafter called a *sandbox*. We use an isolated environment (e.g., a virtual machine) for the dry run, since it can reduce interference from other running applications and ensures better estimates of the performance for the pipelines. The approach enables one to run the pipeline and analyze it in a lower cost environment than simulators, which do additional processing to simulate the actual computing environment like the Cloud or Edge

This is the author accepted version of an article published in
2022 IEEE 46th Annual Computers, Software, and Applications Conference (COMPSAC)
https://doi.org/10.1109/COMPSAC54236.2022.00182

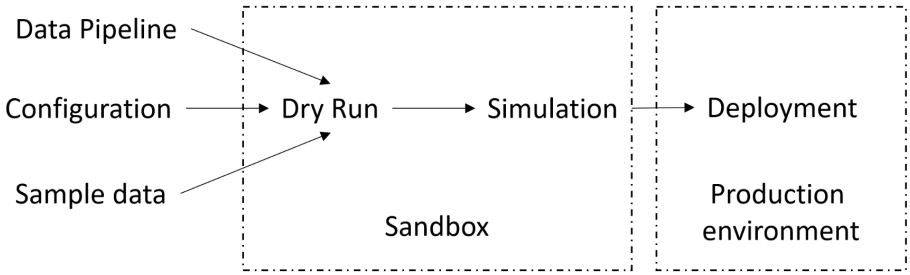

Fig. 1. Dry run approach for testing and simulating big data pipelines.

where it will be deployed in production. The approach, firstly, could be used to check the correctness of the pipeline and to ensure that the pipeline is working as expected and producing the expected output. Secondly, dry run results can be used in simulators to aid in predicting the performance of the pipeline and identify possible bottlenecks. Thereby, the dry run result of the pipeline for a small data size may be used to predict the performance for bigger data sizes, assuming that the data are processed in chunks/slices. For example, metrics collected by dry running with different chunk sizes can be used to estimate infrastructure resources required for scaling the pipeline (e.g, CPU, memory and disk size, and using multiple processes). Software container technologies could simplify the execution of data pipelines [8] both in isolated and production environments by encapsulating individual data pipeline steps in platform and programming language independent containers. In this paper, we describe the proposed dry run approach and present a tool—the SIM-PIPE DryRunner tool—implementing the approach. The overall SIM-PIPE solution aims at using the dry run results for testing the pipelines and simulating them using existing simulators.

The rest of the paper is organized as follows. Section II provides the description of our approach as well as the technical architecture and implementation. In Section III, we present a use case for the proposed approach, while Section IV presents related work. In Section V, we summarize our approach and provide directions for future work.

II. SIM-PIPE DryRunner Approach

The proposed approach based on dry running of big data pipelines relies on the use of an isolated sandbox environment to execute pipeline steps. By maintaining an isolated testing environment, we are able to get an estimate of the resource usage of each step without interference from other running processes. Moreover, the container-based implementation of the step facilitates accurate estimation of its total execution time in the actual deployment infrastructure. This is due to the homogeneity of container technologies, which ensures that the execution of the container is reproducible regardless of the computing infrastructure in which it is executed. Thus, by running the container-based implementations of the pipeline steps, we ensure that we obtain values from dry run, which can be used to predict how the pipeline behaves on resources on the Cloud-Edge continuum.

Figure 2 shows the main steps of the dry run process. Once a dry run is initiated, a step in the pipeline and sample data are deployed to the sandbox using a container. During the execution of the step, execution time will be recorded and the sandbox will be continuously pooled for metrics about the execution. These metrics are stored for later use. Once the step has successfully performed the data processing task, the resulting data will be retrieved, the running step will be removed from the sandbox, and the same process will be repeated for the next steps (i.e., deploy the step and feed it with the resulting data from the previous one). Based on the data gathered, analytics will be performed to derive results that apply to the entire pipeline. The pipeline steps, in case of steps performing batch processing, are provided with a sample input to be used during the dry run. In case of steps which perform continuous processing, there is a user defined option to provide the number of seconds to wait before the step is terminated, this ensures that the correctness of the step and recording of resource usage metrics can be done for that specified amount of time. All the details including resource usage statistics, inputs to the steps, and outputs of the execution are stored and eventually used to perform resource usage analytics.

In the following we describes the technical architecture and implementation of the SIM-PIPE DryRunner tool, and outline a typical use of the tool.

A. Technical Architecture and Implementation

In order to demonstrate the feasibility of the approach for dry running of big data pipelines, we designed and implemented a prototype application—the SIM-PIPE DryRunner tool. It consists of several components that are deployed separately in order to ensure an appropriate execution environment for the dry run approach. The current version of the tool, along with installation instructions are available on GitHub[1].

Figure 3 shows the deployment topology and architecture for SIM-PIPE DryRunner tool. The tool is designed to be deployed in two separate hosts: one for hosting the front-end and business logic, and one for hosting the sandbox environment. The main component is the dry run controller, which performs a step-wise analysis of the pipeline by deploying steps and

[1]https://github.com/DataCloud-project/SIM-PIPE

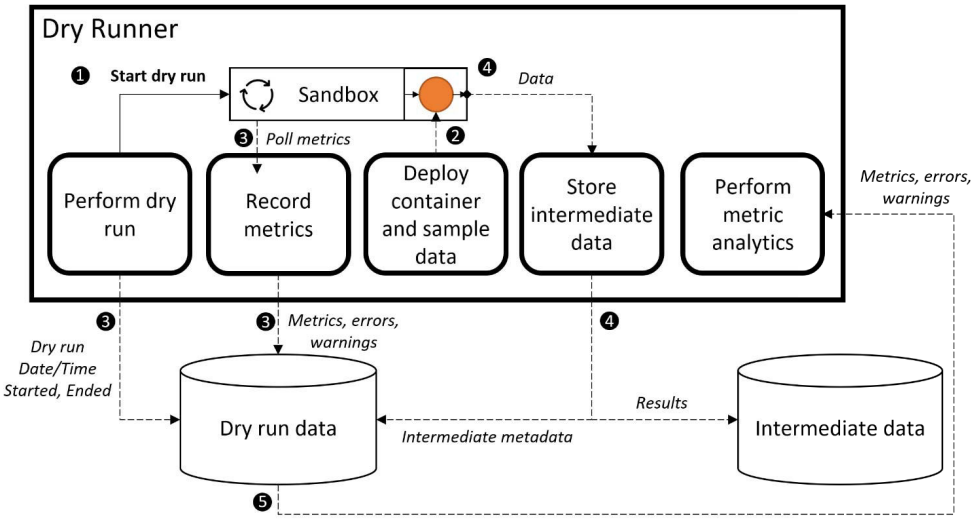

Fig. 2. The SIM-PIPE DryRunner process for testing and collecting performance data.

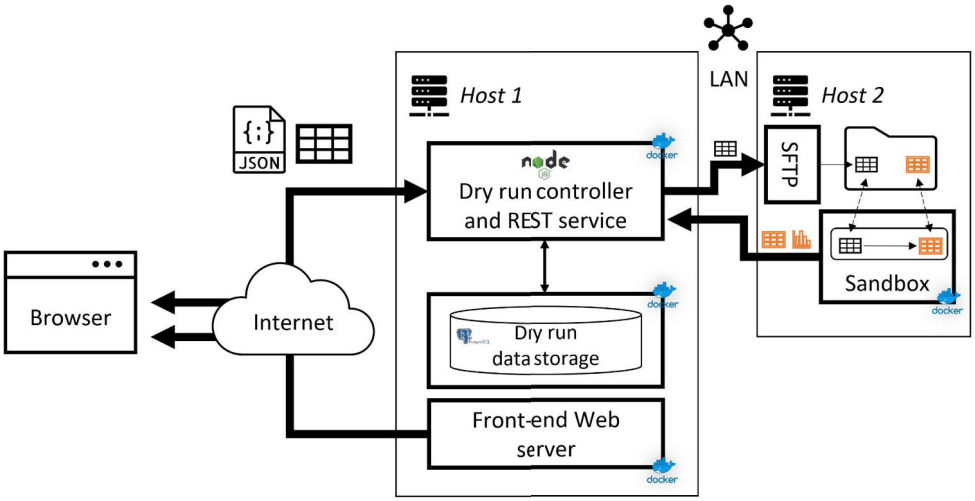

Fig. 3. SIM-PIPE DryRunner tool: deployment topology and architecture.

collecting relevant data. Host 1 in Figure 2 contains the dry run controller and REST service (which serves the front-end of the implementation) as well as the dry run data storage, which is implemented using TimescaleDB[2]. In our implementation, these sub-components are deployed on the host using Docker containers. The necessary files for providing the input and storing the output of each step are transmitted and stored using an SFTP server which also runs in a Docker container in host 2. When deploying a step to be analyzed, the dry run controller sends (if needed) data over SFTP to the sandbox host, which makes it available to the container and executes the step.

The dry run controller and REST service are implemented using NodeJS[3] and use a number of NodeJS libraries related to managing the execution of containers on a target host, namely dockerode[4] for container execution control in the sandbox and ssh2-sftp-client[5] for interacting with the SFTP server on the sandbox. The REST API is developed using GraphQL[6] (a query language for APIs). Hasura[7] is used to develop and connect to the data model of the dry run data storage. The front-end of the SIM-PIPE DryRunner tool is implemented using Appsmith[8].

The current version of the SIM-PIPE DryRunner tool user interface is depicted in Figure 4. The interface displays a list of

[2]https://www.timescale.com
[3]https://nodejs.org
[4]https://github.com/apocas/dockerode
[5]https://github.com/theophilusx/ssh2-sftp-client
[6]https://graphql.org
[7]https://hasura.io
[8]https://www.appsmith.com

dry runs tied with a specific pipeline as well as the associated runs to each dry run. For each run, it displays the run state ("Waiting", "Queued", "Active", "Completed", "Failed", or "Cancelled") as well as statistics on each of the steps. The statistics include the used CPU, memory, network, and running time. In addition to the statistics, the current version of the user interface displays logs from the execution of the steps. The tool assumes that the pipeline description is provided in the form of a Domain Specific Language (DSL) which is described in a Github repository[9]. This DSL has been developed as part of the DEF-PIPE tool which is a GUI (Graphical user Interface) based tool to design, implement and store big data pipelines. More details and usage guidelines of this tool are given in a Github repository[10].

The current implementation supports explicitly step implementations as described in the big data pipeline approach in [9], whereby each container collects input data, stores output data, and any intermediate data separately in a file system. Thereby, the SIM-PIPE DryRunner tool provides input data to the steps and stores intermediate step outputs for analysing the dry run. Other step implementations that do not use file-based data transmission are also applicable, but the data delivery system currently does not support this.

The dry run data storage uses a relational database model and records each dry run with a timestamp and pipeline identifier. Each run is also associated with the DSL model that was used when the run was started as well as its (current) status and the timestamps when the run was created, started, and ended. Each run stores data for each of the steps that are in the input DSL model with the step name, status, and metrics about the used CPU and memory. Intermediate data are stored on disk in a file system that are marked with the pipeline identifier, run identifier, and step number and can be served on request to the front-end.

B. Using the SIM-PIPE DryRunner tool

Dry run using the SIM-PIPE DryRunner tool is done through the following steps:

- First, the user creates a new dry run for a pipeline by providing its DSL description and sample input data using the SIM-PIPE DryRunner tool UI.
- The user starts a new dry run and the current status of the run and each step is displayed in the UI.
- After each step has completed execution indicated by its status, the user can click on the step to view the logs generated during execution, CPU usage percentage, network usage, memory usage and maximum memory usage over time.
- In case of failure of a step, the status of the step and correspondingly run would indicate failure status, and only the logs would be displayed which may help in debugging.
- The step can also be stopped while running, and this stops the current step and all the succeeding steps in the pipeline.

[9]https://github.com/DataCloud-project/DEF-PIPE-DSL
[10]https://github.com/DataCloud-project/DEF-PIPE

III. Use Case

The SIM-PIPE DryRunner tool was tested on data pipelines in the context of a digital health system, where developers and data engineers are using data pipelines to implement different e-health services. The main objective of the digital health system is to monitor, support and help patients, especially elderly, at their homes, remotely. The system uses data pipelines to gather sensor data (e.g., welfare sensors and medical devices) from the patients, store and process the patient data, and provide relevant data to the right stakeholder at the right time (e.g., notifications of events to healthcare providers, storing data in electronic health records, and providing data and notifications to third party health systems).

Figure 5 illustrates a generic digital health data pipeline that involves three steps: 1) Data generation, pre-processing and routing, 2) Data storage and analysis, and 3) End user application logic. The first step is deployed on the Edge, while the two latter are deployed on the Cloud. The steps are the same three steps shown in the SIM-PIPE DryRunner tool UI in Figure 4. The first step involves collecting and formatting sensor data from healthcare sensors and medical devices that the patient uses. The second step involves storing the data and checking it against the patient plan. The third step involves different types of end user application logic, such as notifying healthcare providers and submitting reports to 3rd party healthcare systems.

Several instances and variants of data pipelines are deployed in the digital health use case. There are pipeline instances for each patient. Some of the challenges in managing the various variants of pipelines relates to i) scaling individual steps of the pipeline, ii) the need to build new applications for each new type of sensor, and iii) finding the optimal resource allocation for data processing steps. The SIM-PIPE DryRunner tool is used to address these challenges, allowing the developers and data engineers of the digital health data pipelines to test new variants of the pipelines without deployment on production infrastructure in order to identify trouble spots and bottlenecks early, as well as better understand the resource requirements required from the metrics collected by the SIM-PIPE DryRunner tool.

IV. Related work

There are several simulation approaches for data pipelines that include tools to simulate big data pipelines, such as the event-based simulator GroudSim [5], and process-based simulators GridSim [6] and CloudSim [7]. Despite the number of simulation approaches in literature, there are few that can be used for testing and simulation of big data pipelines. Liu et al. [10] present a survey of scientific workflow management systems in the context of big data pipelines, out of the five

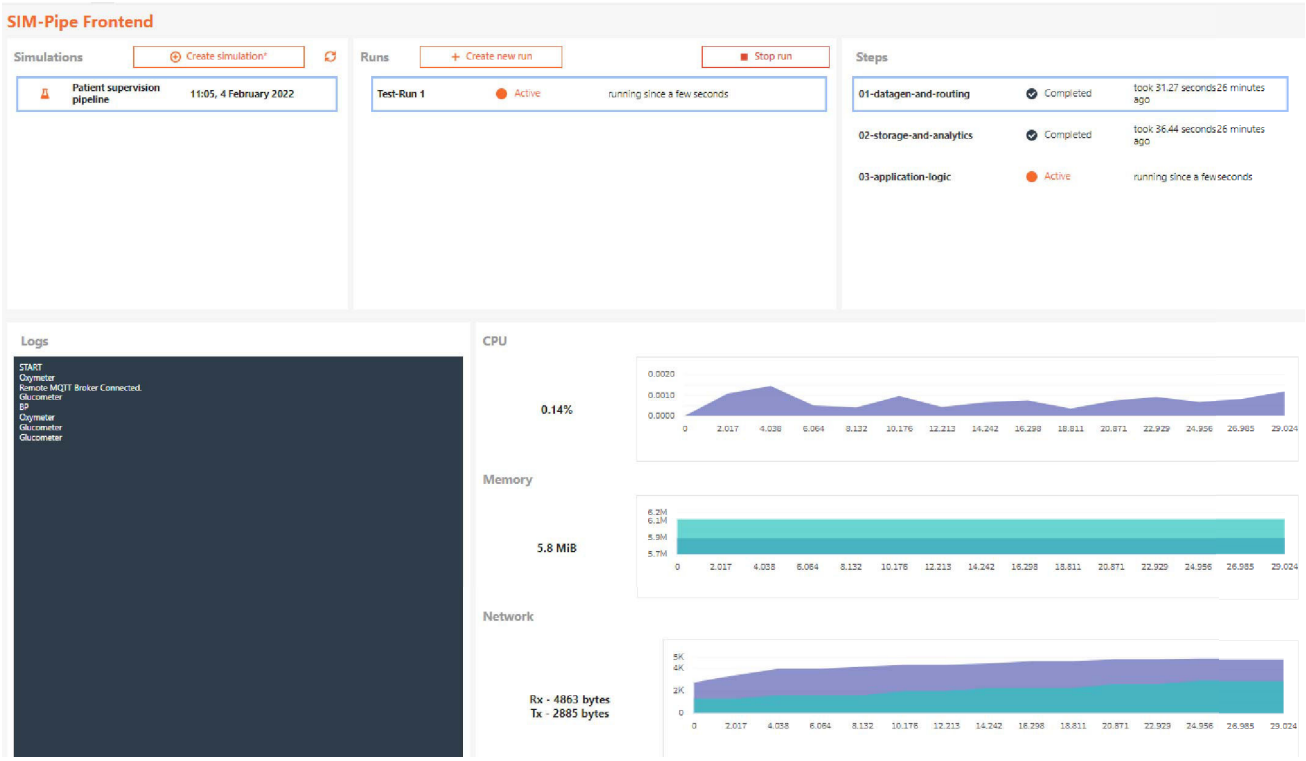

Fig. 4. SIM-PIPE DryRunner tool front-end.

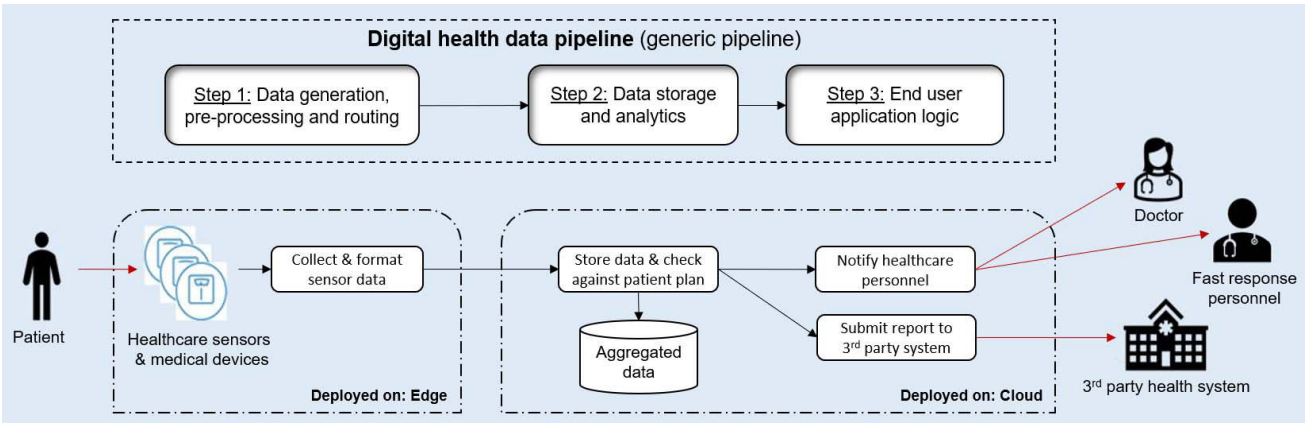

Fig. 5. SIM-PIPE DryRunner tool front-end.

systems presented only two of them (Taverna[11], Swift[12]) had a simulation or testing component. While Taverna is specialized to support bio-informatics pipelines, Swift only provides tools for unit and integration testing of pipelines. These simulators vary in ways in which they accept data for simulating a pipeline. Many of them run pipelines multiple times and the results from the runs are used in simulation [11].

Iatropoulou et al. [12] present a data pipeline management system for container-based big data pipelines and supports design, composition, configuration, orchestration, enactment, and validation of end-to-end big data analytic services. Each step in the input pipeline is provided in the form of one of the four predefined containerized application images (named as Apps) which is part of their microservices architecture. Though it handles several types of big data workflows, it is not open source and thus cannot be extended.

[11] https://incubator.apache.org/projects/taverna.html
[12] https://github.com/square/workflow-swift

V. Conclusions and Outlook

We proposed a new approach—SIM-PIPE DryRunner—for dry running of big data pipelines using an isolated sandbox for deployment of steps. Testing and simulation of big data pipelines is challenging, since the existing methods depend on information from previous runs or domain expert knowledge, which are difficult to acquire in case of big data pipelines. We also developed an initial version of the tool—the SIM-PIPE DryRunner tool—with a user interface in which the pipeline designer can input and dry run big data pipelines and view the results of the resource usage of step execution and logs. The dry run results of the big data pipeline can be used in existing simulators by bringing them into the respective format that can be used as input. One limitation of this method is that it assumes that the big data pipelines have container-based implementations.

In the future, we aim to enable the SIM-PIPE DryRunner tool to recommend minimum requirements for the resources necessary to run the pipeline steps successfully (i.e., the minimum memory and CPU requirements) and to provide an estimation of the optimal horizontal scaling for each individual step that will allow for executing the pipeline without bottlenecks. Future work also involves extending it further by integrating advanced analytics for the results obtained from the sandbox. This involves predicting the resource usage performance and total execution time of the pipeline when a given input size is specified. We also aim to analyze and quantify the impact of parallelisms for various pipeline steps. This can be used in configuring the resources at deployment or in scheduling algorithms. Finally, we also plan to use the dry run results in existing simulators. This requires investigation of input formats which is accepted by these simulators and conversion of the output of our tool into a format that is usable by them.

Acknowledgements. This work received partial funding from the European Commission Horizon 2020 DataCloud project (grant number 101016835), the NFR BigDataMine project (grant number 309691), and the SINTEF internally funded SEP DataPipes project.

References

[1] R. Buyya, S. N. Srirama, G. Casale, R. Calheiros, Y. Simmhan, B. Varghese, E. Gelenbe, B. Javadi, L. M. Vaquero, M. A. S. Netto, A. N. Toosi, M. A. Rodriguez, I. M. Llorente, S. D. C. D. Vimercati, P. Samarati, D. Milojicic, C. Varela, R. Bahsoon, M. D. D. Assuncao, O. Rana, W. Zhou, H. Jin, W. Gentzsch, A. Y. Zomaya, and H. Shen, "A manifesto for future generation cloud computing: Research directions for the next decade," *ACM Computing Surveys*, vol. 51, no. 5, 2018.

[2] M. Barika, S. Garg, A. Y. Zomaya, L. Wang, A. V. Moorsel, and R. Ranjan, "Orchestrating big data analysis workflows in the cloud: Research challenges, survey, and future directions," *ACM Computing Surveys*, vol. 52, no. 5, 2019.

[3] A. Shakarami, H. Shakarami, M. Ghobaei-Arani, E. Nikougoftar, and M. Faraji-Mehmandar, "Resource provisioning in edge/fog computing: A comprehensive and systematic review," *Journal of Systems Architecture*, vol. 122, p. 102362, 2022.

[4] I. Bambrik, "A survey on cloud computing simulation and modeling," *SN Computer Science*, vol. 1, no. 5, p. 249, 2020.

[5] S. Ostermann, K. Plankensteiner, R. Prodan, and T. Fahringer, "Groudsim: An event-based simulation framework for computational grids and clouds," in *Proceedings of the Euro-Par Parallel Processing Workshops (Euro-Par 2020)*, ser. LNCS, vol. 6586. Springer, 2010, pp. 305–313.

[6] R. Buyya and M. Murshed, "Gridsim: A toolkit for the modeling and simulation of distributed resource management and scheduling for grid computing," *Concurrency and computation: practice and experience*, vol. 14, no. 13-15, pp. 1175–1220, 2002.

[7] R. N. Calheiros, R. Ranjan, A. Beloglazov, C. A. De Rose, and R. Buyya, "Cloudsim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms," *Software: Practice and experience*, vol. 41, no. 1, pp. 23–50, 2011.

[8] M. Matskin, S. Tahmasebi, A. Layegh, A. H. Payberah, A. Thomas, N. Nikolov, and D. Roman, "A survey of big data pipeline orchestration tools from the perspective of the datacloud project," vol. 3036, 2021.

[9] N. Nikolov, Y. D. Dessalk, A. Q. Khan, A. Soylu, M. Matskin, A. H. Payberah, and D. Roman, "Conceptualization and scalable execution of big data workflows using domain-specific languages and software containers," *Internet of Things*, vol. 16, p. 100440, 2021.

[10] J. Liu, S. Lu, and D. Che, "A survey of modern scientific workflow scheduling algorithms and systems in the era of big data," in *Proceedings of the IEEE International Conference on Services Computing (SCC 2020)*. IEEE, 2020, pp. 132–141.

[11] T.-P. Pham, J. J. Durillo, and T. Fahringer, "Predicting workflow task execution time in the cloud using a two-stage machine learning approach," *IEEE Transactions on Cloud Computing*, vol. 8, no. 1, pp. 256–268, 2017.

[12] S. Iatropoulou, P. Petrou, S. Karagiorgou, and D. Alexandrou, "Towards platform-agnostic and autonomous orchestration of big data services," in *Proceedings of the IEEE Seventh International Conference on Big Data Computing Service and Applications (BigDataService 2021)*. IEEE, 2021, pp. 1–8.