DOI: 10.14529/jsfi210106

Micro-Workflows Data Stream Processing Model for Industrial Internet of Things

Ameer B. A. Alaasam[1], ***Gleb I. Radchenko***[1], ***Andrei N. Tchernykh***[1,2,3]

The fog computing paradigm has become prominent in stream processing for IoT systems where cloud computing struggles from high latency challenges. It enables the deployment of computational resources between the edge and cloud layers and helps to resolve constraints, primarily due to the need to react in real-time to state changes, improve the locality of data storage, and overcome external communication channels' limitations. There is an urgent need for tools and platforms to model, implement, manage, and monitor complex fog computing workflows. Traditional scientific workflow management systems (SWMSs) provide modularity and flexibility to design, execute, and monitor complex computational workflows used in smart industry applications. However, they are mainly focused on batch execution of jobs consisting of tightly coupled tasks. Integrating data streams into SWMSs of IoT systems is challenging. We proposed a micro-workflow model to redesign the monolith architecture of workflow systems into a set of smaller and independent workflows that support stream processing. Micro-workflow is an independent data stream processing service that can be deployed on different layers of the fog computing environment. To validate the feasibility and practicability of the micro-workflow refactoring, we provide intensive experimental analysis evaluating the interval between sensor messages, the time interval required to create a message, between sending sensor message and receiving the message in SWMS, including data serialization, network latency, etc. We show that the proposed decoupling support of the independence of implementation, execution, development, maintenance, and cross-platform deployment, where each micro-workflow becomes a standalone computational unit, is a suitable mechanism for IoT stream processing.

Keywords: stream processing, fog computing, cloud computing, scientific workflow, micro-workflow, IoT.

Introduction

The Industrial Internet of Things (IIoT) comprises networked objects, cyber-physical assets, associated information technologies, cloud and edge computing platforms. It enables real-time data processing and exchange of processes, products, and service information within the industrial environment to optimize overall production value [7]. An essential feature of computing services in IIoT are on-site processing, ensuring security requirements, and data pre-processing before sending it to the cloud. Thus, there is a need for an intermediate processing power between industrial IoT and cloud [1]. Fog computing provides such resources by moving some of the data processing tasks from the cloud to fog nodes located closer to the network's edge.

IIoT applications require real-time processing of data streams and signals from multiple sensors (data sources). Event-Driven Architecture (EDA) is the most adapted to this type of applications [3]. EDA is a system architecture made up of highly decoupled, single-purpose event processing components which asynchronously receive and process events [28]. EDA, by its nature, is extremely loosely coupled and highly distributed [12]. A monolithic architecture does not provide the efficiency and flexibility required to support large-scale IIoT systems that require native EDA support and a multilayered fog computing infrastructure.

[1]South Ural State University, Chelyabinsk, Russia
[2]CICESE Research Center Ensenada, Mexico
[3]Ivannikov Institute for System Programming of the RAS, Russia

Due to the complexity in controlling a distributed application workflows, Workflow Management Systems (WFMSs) are often used to assist in the data and task partitioning, providing robust means of describing applications, the control, data dependencies, and the logical reasoning necessary for distributed execution [31]. But a number of challenges appear with WFMSs, for example, the tasks of the workflow are tightly coupled by means of intricate dependencies between them [35]. Also, the features of data visualization and data stream input/output are limited support in current WFMSs [5]. Thus in [2, 27], the concept of Micro-Workflows has been presented. The Micro-Workflow approach supports the redesign of monolith workflows into independent smaller workflows, maintaining stream processing and independent lifecycle management.

This paper presents the micro-workflows model supporting the decoupling process of tightly coupled dependencies in monolith workflow to be refactored in the form of independent smaller micro-workflows, connected via event streaming platform. The Micro-Workflow model can solve a number of problems when using traditional workflow management systems in highly distributed environments, such as fog computing systems to support IIoT. Also, we provide an overview of current IIoT and fog computing challenges in areas such as monolithic application architecture, virtualization, containerization, computational workflows, and data flow management.

The rest of this paper is organized as follows. Section 1 discusses the road from a monolithic into a loosely coupled system architecture, the challenges of fog and cloud computing, the required virtualization infrastructure, and discusses the importance and challenges of computational workflows and data flow management in fog environments. Section 2 provides the proposed model of the micro-workflow architecture. Section 3 provides the experiments and results, while conclusions are provided in Section 3.3.

1. The State of the Art

1.1. From Monolith to Loosely-Coupled Architecture

An event-driven architecture includes sensors and other sources of data; processors that fuse data from multiple sensors and detect patterns over time and deduce events that occurred or predict events; responders for initiating actions in response to events; communication links for transferring information between components; and administrative software for monitoring, tailoring and managing the application [10]. EDA is an extremely loosely coupled system architecture that is made up of highly decoupled, single-purpose event processing components that asynchronously receive and process events [12, 28]. Microservice architecture is considered the most promising in designing loosely coupled systems targeted at event processing today. The microservice approach is based on dividing a computing system into small independent computing services, each implementing its particular aspect of the application's business logic. This approach can overcome many of the significant disadvantages of the so-called monolithic architecture, such as the challenges of distributing the computational load, the presence of a single point of failure, and the challenges of ensuring continuous system operation during maintenance or upgrades while allowing independent development, deployment, scaling and migration of microservices from one computational resource to another [29]. Despite the benefits of this approach, such flexibility does not come for free. For example, communication costs increase due to the need to organize data exchange between microservices, which, in turn, increases the complexity of data flow management and integration of highly distributed components of the

system. Thus, moving to a more loosely coupled design is a multi-objective problem that requires in-depth research to find the best solution to the various issues that arise from decoupling.

1.2. From Cloud to Fog Computing

In highly distributed industrial IoT systems, cloud computing-based solutions become unacceptable due to high latency and network congestion caused by an inability to process the data-flow from the enormous amount of devices in an acceptable time [21]. Thus, Fog computing manages this problem by moving some computational tasks closer to the data sources. Fog Computing is a virtualized platform that provides storage, compute, and networking between end devices and Cloud Computing Data Centers, typically, but not exclusively, located at the edge of the network [6]. Fog nodes can collect, cache, and pre-process data from IIoT sensors before or instead of sending it to the cloud. Another scenario may require certain parts of the system to process data in near-real time. In this case, computing services can be deployed on the nearest fog node to provide a faster response time.

On the other hand, Cloud servers provide significant replication, load balancing, and resilience capabilities. Fog nodes, which are logically decentralized and geographically distributed at network edges, cannot provide this resilience level. For example, finding and resolving a point of failure in a fog computing environment is more complicated than doing that in the cloud [14]. Also, moving toward fog means increasing the computing decoupling, where parts of computing are moved physically to the fog nodes at the edge of the network. Increased component decoupling can lead to difficulties in supporting such systems. For example, the emergence of a large number of independent geographically decoupled components entails significant overhead due to the appearance of a large number of points of failure [18]. The problems of efficient workload allocation, the distribution of computational tasks between edge and cloud resources, and the heterogeneous infrastructure of edge devices need to be solved [8].

1.3. Virtualization Infrastructure

In cloud or fog computing, ensuring resource sharing and dynamic resource provisioning is a fundamental challenge. Virtualization technologies used at various levels (including hardware and application platforms) are used for this purpose. However, large overheads associated with the use of Virtual machines (VMs) can limit the efficiency of computational resources [34]. The challenges associated with using fog-level VMs are even more significant due to the limited resources, processing power, and network traffic at the edge of the network where fog nodes are deployed. This problem can be addressed by containerization technology which allows running containers as separate processes directly on the kernel of hosting OS, providing lightweight isolation for the processes. However, using containerization also facing significant challenges related to the difficulty of containerizing the stateful computational units due to limited data portability support in containers compared to VM [4]. Therefore, if live migration between fog nodes is required, it is a challenge to use container technology without data loss. Thus, finding solutions that reduce the overhead of the virtualization infrastructure, and at the same time ensuring a level of data availability and state recovering, is a very active and vital research area in fog computing.

1.4. The Computational Workflow

The Digital Twin concept combines a control system, real-time simulation, a system for intelligent data analysis, and decision-making when developing industrial processes and systems [26, 32]. This is made possible by data flow and control signals connecting real-world objects and their digital models. Experience in implementing such digital twins shows that the simulation of complex technological processes requires the joint work of a large number of independent computing components, each of which is responsible for implementing its part of the computational process. Such an approach in the field of the smart industry is already implemented using scientific workflow systems. For example, in [20], a specific workflow suit has been developed for product performance degradation assessment and prediction based on Kepler scientific workflow management system (Kepler MS). The authors of [19] used Kepler in smart manufacturing to optimize temperatures across a commercial industrial scale furnace in the steam methane reforming process used to manufacture hydrogen gas. They used Kepler workflows to combine MATLAB and ANSYS packages to manage Computational Fluid Dynamics (CFD) calculations. For distributed computing, additional instances for CFD calculations may be deployed when running in parallel using the MPI message-passing scheme.

Scientific workflows (SWF) are a cornerstone of modern scientific computing. They are used for complex computational applications that require robust management for big data that are typically stored and processed at heterogeneous, distributed resources [30]. SWF systems make scientists focus on their research rather than on details of computation management. For example, the Pegasus framework allows users to represent workflows at an abstract level while it takes care of the execution systems particulars [11]. Similarly, the main goal of the Kepler MS is to support different execution scenarios where a user can develop and use its modules to manage different execution behaviors in different environments, including private computational resources [25]. The unique features of Kepler are that the underlying workflow engine handles the provenance, reproducibility aspects of the code, performs orchestration of data flow, and automates execution on heterogeneous computing resources [33].

However, several issues arise regarding the use of the WFMs. Workflows are executed as a batch processing model, where a set of data is collected and fed into a workflow as a batch, which is processed sequentially within the corresponding workflow [15]. The workflow tasks, in this case, are closely related to each other due to the complex dependencies between them. Also, workflow jobs may generate a large amount of intermediate data during the workflow lifecycle [35]. In such tightly coupled behavior, a heavy data transfer among workflow tasks can cause a significant slow down in execution [22]. Thus, to manage and efficiently execute workflows, it is necessary to consider the features of this type of computing process, including the limitation of resources over time and planning features, taking into account the location of resulting and intermediate data [16]. One solution for such a monolith behavior problem is to divide it into smaller elements. For example, the authors in [17] proposed to divide the problem of workflow into two smaller subproblems; the first to allocating multiple workflows into multiple data centers, and the second for allocating the tasks of each workflow into the computing resources inside each data center. However, still, each traditional workflow working in batch execution mode over tightly coupled tasks. Another challenge is that the features of data visualization and data stream input/output are limited support in current WFMSs [5]. Such complexity increased in case of fog computing systems, where the execution environment itself decoupled over multiple separated geographical locations. Thus, to fit fog computing EDA

requirements, the computing system itself should be decoupled into fine-grained services that support independent deployment and lifetime management.

1.5. The Data Flow

When organizing the computational process for systems such as the Digital Twin, the data management process organization is essential. In addition to fundamental factors such as volume and data type, several key factors influence the complexity of the data management process organization related to the properties of the computing environment that supports Digital Twin.

First, the digital twin requires combining stream data processing from sensors with batch data processing when performing intelligent analysis or big data processing tasks. Second, the multi-layered and geographically distributed nature of the underlying fog computing environment requires the organization of the data flow management system as a distributed system consisting of independent components, the communication between which is organized through message exchange. Moreover, data processing for industrial applications often requires the ability to predict the state of the system. Such a service needs to know information about state history to perform prediction. This behavior is called stateful, which means that the system must identify the input data source and determine what other input data came from the same source [24]. This requirement makes the lifetime management of services that are responsible for data flow processing considerably more challenging.

However, managing state data inside a computational service itself is considered a bottleneck, so it should be stored in a separate resource [23]. An example of such a concept is presented in [9], where the authors proposed a three-layered IoT data workload processing architecture consisting of two layers for data management (messaging layer for data input/output and volatile layer - to cache the updated results), while all the processing is concentrated in the third layer. In the other case [13], the authors built a digital twin system, where a central data layer based on a publish-subscribe architecture using MQTT messaging protocol broker linked together a physical object, a digital twin, and a web-based controlling dashboard integrated with the CAD system.

2. Micro-Workflow Model

In works [2, 27], the concept of Micro-Workflow was presented in an abstract form. The current work expands this concept to be an in-detail representation with the details of creating a Micro-Workflows oriented from the partitioned workflow. Also, it defines the inputs and outputs and their accurate representation, how this model helps solve a set of challenges in using workflow in fog computing.

The Micro-Workflow concept supports redesigning a monolithic workflow into a set of smaller, loosely-coupled Micro-Workflows (MWF). Each MWF acts as an independent service supporting stream data processing, independent deployment, and communication via the streaming middleware.

2.1. The Monolith Workflow

A workflow application can be represented as a Directed Acyclic Graph (DAG):

$$W = (V, E), \tag{1}$$

where:

W – the monolith workflow;

V – a set of vertices representing computational tasks;

E – a set of directed edges connecting vertices.

Each vertex v_i in V can have input and output edges. Let's define the in-degree and out-degree of a vertex v_i in W by $deg^+(v_i)$ and $deg^-(v_i)$ respectively. Let the vertex v_1 in V be the initial vertex of the W that has no predecessors. Let n be the total number of vertices in V, so v_n would be the final vertex that does not have any successor tasks. An edge $(v_i, v_j) \in E$ represents the data that is going as one of the outputs produced by v_i to be as one of the inputs of v_j. Figure 1 shows an example of workflow W where n assumed to be 5.

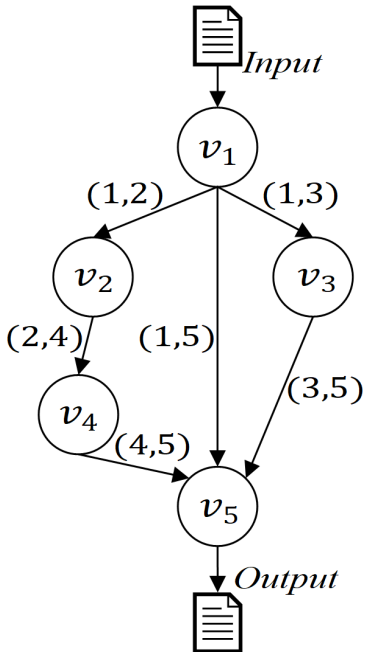

Figure 1. An example of workflow W where n assumed to be 5

2.2. The Sub-Workflow

We say that the workflow W is divided into a set of sub-workflows $S = (S_1, \dots S_k)$, $S_i = (V_i, E_i)$, when:

1. $\forall i = 1 \dots k \;\; (V_i \subset V, E_i \subset E)$;
2. $\forall v \in V, \exists S_i \;\; (v \in V_i)$;
3. $\forall i, j \;\; (i \neq j \implies S_i \cap S_j = \emptyset)$.

Figure 2 illustrates an example of workflow W partitioned into two sub-workflows S_1 and S_2.

2.3. Sub-Workflow Edges Classification

Let us define the following classes of edges and vertices, associated with the sub-workflow S_i:

- $EINT_i$: a set of internal edges, located inside of the sub-workflow S_i:

$$EINT_i = \{(v_k, v_l) \in E : v_k, v_l \in S_i\} \tag{2}$$

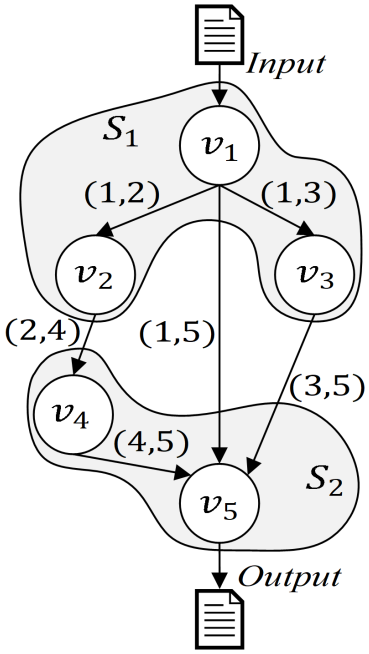

Figure 2. An example of workflow W partitioned into two sub-workflows S_1 and S_2

- $EINPUT_x$: a set of input edges with an initial vertex outside the sub-workflow S_i and an end vertex inside S_i:

$$EINPUT_i = \{(v_k, v_l) \in E : v_k \notin S_i,\ v_l \in S_i\} \tag{3}$$

- $EOUT_i$: a set of output edges with an initial vertex inside the sub-workflow S_i and an end vertex outside S_i:

$$EOUT_i = \{(v_k, v_l) \in E : v_k \in S_i,\ v_l \notin S_i\} \tag{4}$$

- $VINPUT_i$: a set of vertices in S_i that located on the head end of $EINPUT_i$ edges as well as vertices that have no input edges:

$$VINPUT_i = \{v_l \in S_i : (v_k, v_l) \in EINPUT_i\} \cup \left\{v \in S_i : deg^-(v) = 0\right\} \tag{5}$$

- $VOUT_i$: a set of vertices in S_i that located on the tail end of $EOUT_i$ edges as well as vertices that have no output edges:

$$VOUT_i = \{v_k \in S_i : (v_k, v_l) \in EOUT_i\} \cup \left\{v \in S_i : deg^+(v) = 0\right\} \tag{6}$$

2.4. Micro-Workflow Construction

To convert a sub-workflow S_i into a micro-workflow MWF_i, we need to extract all S_i vertices from the workflow W and provide communication mechanisms linking MWF_i with the event streaming platform via dedicated "Consumer vertex" (cv_i) and "Producer vertex" (pv_i) nodes. The cv_i vertex acts as a source of MWF_i, providing consumption of the input data stream from the event streaming platform and distributing it between the vertices in $VINPUT_i$. The pv_i vertex acts as a sink that collects the output from the $VOUT_i$ set and transmits it as a message to the event streaming platform. In this case, we can define the generation of $MWF_i = (MV_i,\ ME_i)$ from the sub-workflow $S_i = (V_i,\ E_i)$ as follows:

1. $MV_i = V_i \cup \{cv_i,\ pv_i\}$;

2. $ECV_i = \{(cv_i, v) : v \in VINPUT_i\}$;
3. $EPV_i = \{(v, cp_i) : v \in VOUT_i\}$;
4. $ME_i = EINT_i \cup ECV_i \cup EPV_i$.

Figure 3 shows the set of micro-workflows produced through the extraction of sub-workflows of the monolithic workflow W shown in Fig. 2.

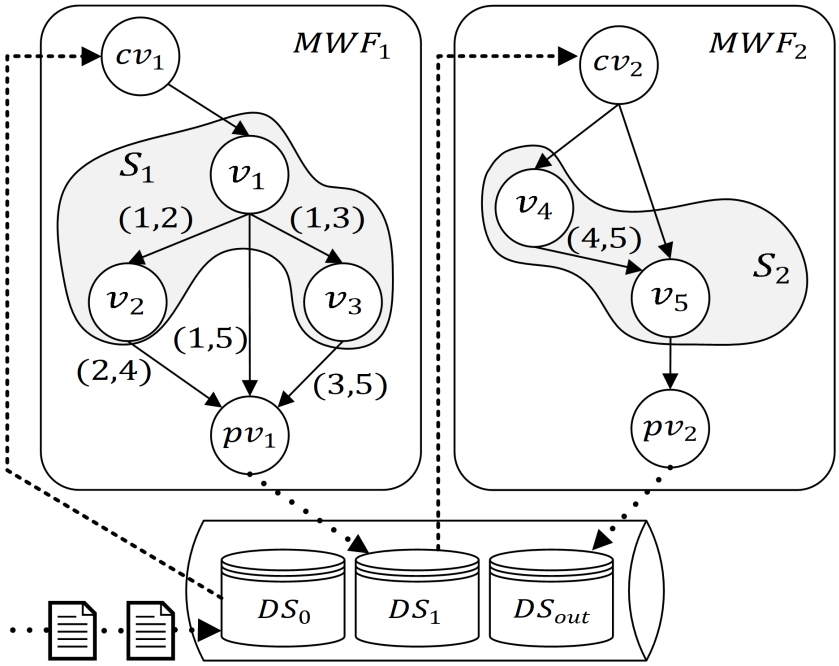

Figure 3. The result of applying the Micro-Workflow model

2.4.1. Implementation of data stream processing in micro-workflows

Micro-workflow concept integrates classical workflow and streaming data processing models. A set of dedicated channels (data stream stores) is formed in the event streaming platform structure to organize the interaction of micro-workflows via message exchange. Each message is a data set that includes a timestamp of message generation, information about the data source, and a structured collection of essential data itself. The following data stream stores are required:

- DS_0 is responsible for collecting, storing, and providing messages that contain the data sets necessary to initialize the computational process in the workflow.
- DS_i is responsible for receiving messages containing intermediate data from MWF_i and passing them to dependent micro workflows.
- DS_{out} is responsible for collecting, storing, and providing messages containing workflow result data.

Instead of the initial workflow node, the data sets needed to start the computational process are fed into the DS_0 data stream store of the event streaming platform in the form of messages. The processing of messages from the data stream store is organized as follows.

- CV_i of the corresponding MWF_i retrieves the subsequent message from DS_{i-1}.
- Based on the analysis of the received message, CV_i generates data transfer along the ECV_i edges to the nodes responsible for the direct execution of the computational process.
- After data processing tasks have been completed and data has been sent along the edges of EPV_i, PV_i generates an outgoing message to DS_i or DS_{out} if it is the final result.

Using this approach to partition a monolithic workflow into a set of independent micro-workflow computing services brings the following advantages:

- decoupling a strongly coupled computational process in time and space, switching to an asynchronous communication model;
- supporting independent micro-workflows deployment on different nodes in a distributed computing environment, taking into account the data source's geographical location, the required computing resources, etc.
- the possibility of seamless integration of IoT device data into the data processing process at any stage of the computing process;
- the ability to independently scale individual micro-workflows;
- the transfer of the computational process of the micro-workflow to a different computational node, without loss of intermediate data, and the need to repeat previous data processing.

3. Experiments

To evaluate the refactoring practicability and possible overhead of dividing the workflow into micro-workflows, we consider data processing from a typical DEBS 2012 data challenge[4] task. As part of this task, it is necessary to provide near-real-time processing of industrial Internet of Things data that transmits information about the state of manufacturing equipment.

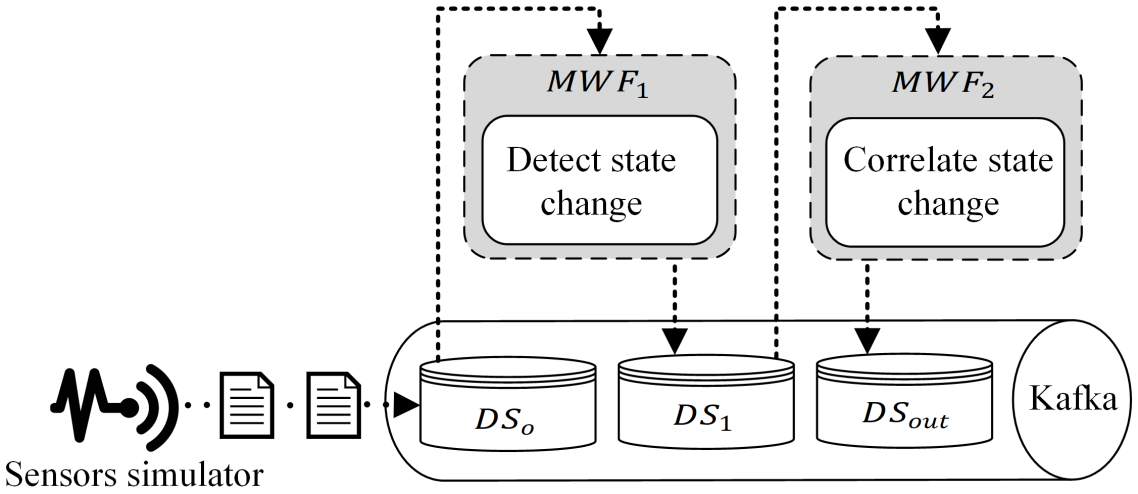

Figure 4. The organization of the experiment

Based on the proposed micro-workflow approach and model, the DEBS 2012 Query 1 monolith workflow has been refactored into two micro-workflows. Each of the micro-workflows has been implemented using the Kepler workflow system and has been packed in a separate Docker container. We have used Apache Kafka[5] as an event streaming platform to organize the message exchange between the micro-workflows. The data stream stores have been implemented as Kafka topics. The organization of the experiment is presented on the Fig. 4. To organize the data exchange between the Kepler workflow and the Apache Kafka platform, we have implemented specialized Kepler actors: KafkaConsumer, which acts as a consumer vertex, and KafkaProducer, which acts as a producer vertex for corresponding micro-workflows. We have

[4]https://debs.org/grand-challenges
[5]https://kafka.apache.org/

also developed the sensors simulator that feeds the initial sensors data stream into the DS_0 topic of the Apache Kafka.

The sensor simulator receives data from the pre-recorded sensor readings database, serializes the messages in a predetermined format, and transmits them to the DS_0 Kafka topic. Before generating and sending the following message in both experiments, we add a delay of 8 ms after receiving a successful message reception confirmation from Apache Kafka. On the one hand, this allows to approximate the data generation frequency to the one determined in the initial data stream (about 10 ms). On the other hand, we are able to keep the exact value of the introduced delay in both experiments.

The structure of the first micro-workflow MWF_1 is presented on the Fig. 5. MWF_1 is a micro-workflow that consumes the sensor data from the DS_0 Kafka topic and processes it using the *DetectStateChange* Kepler actor. The data processing results are published to the intermediate Kafka topic DS_1.

The structure of the second micro-workflow MWF_2 is presented on the Fig. 6. MWF_2 consumes the data from the DS_1 Kafka topic. It implements the second stage of data processing, including the correlation of the change of state of the sensor and the change of state of the valve by calculating the time difference between the occurrence of the state changes. The correlation estimation is performed in the *CorrelateStateChange* Kepler Actor.

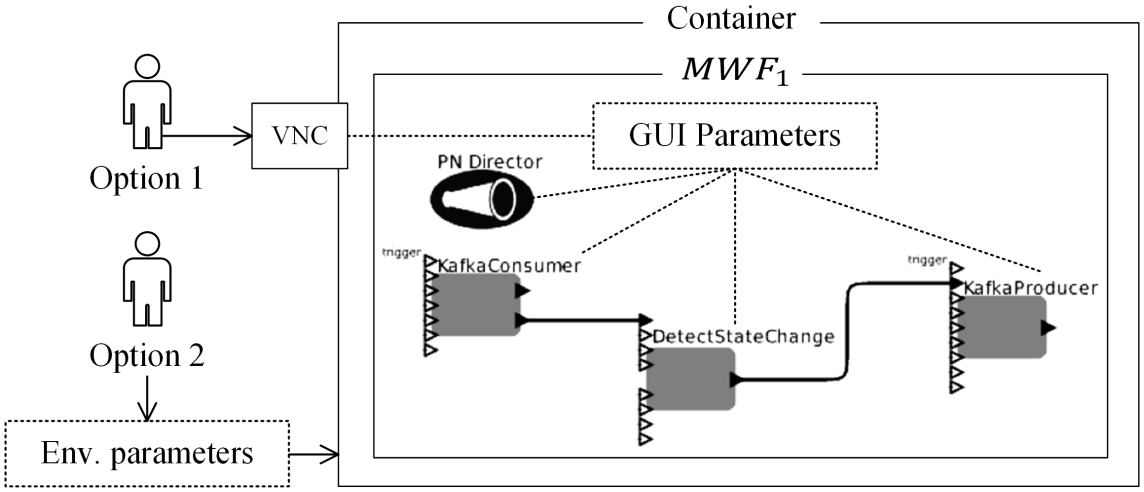

Figure 5. The two options of running MWF_1 in the developed Kepler model

3.1. Micro-Workflow Deployment Parameterization

Before launching the micro-workflow, it is necessary to configure its parameters, including the information needed to communicate with the outside world. The micro-workflow should be provided with information about the location of the endpoint address of the event streaming platform (Apache Kafka in our case), the location of the message schema repositories, and the identifiers of the data stream stores for reading and writing messages.

In our previous work [27], we have implemented the parameterization using remote desktop access to the GUI of each workflow and manually enter all execution parameters to the workflow. In this paper, we improve the implementation of the micro-workflow deployment model using the headless option. In this case, micro-workflow parameters can be provided as execution parameters into the docker container at the micro-workflow container's launch. The execution

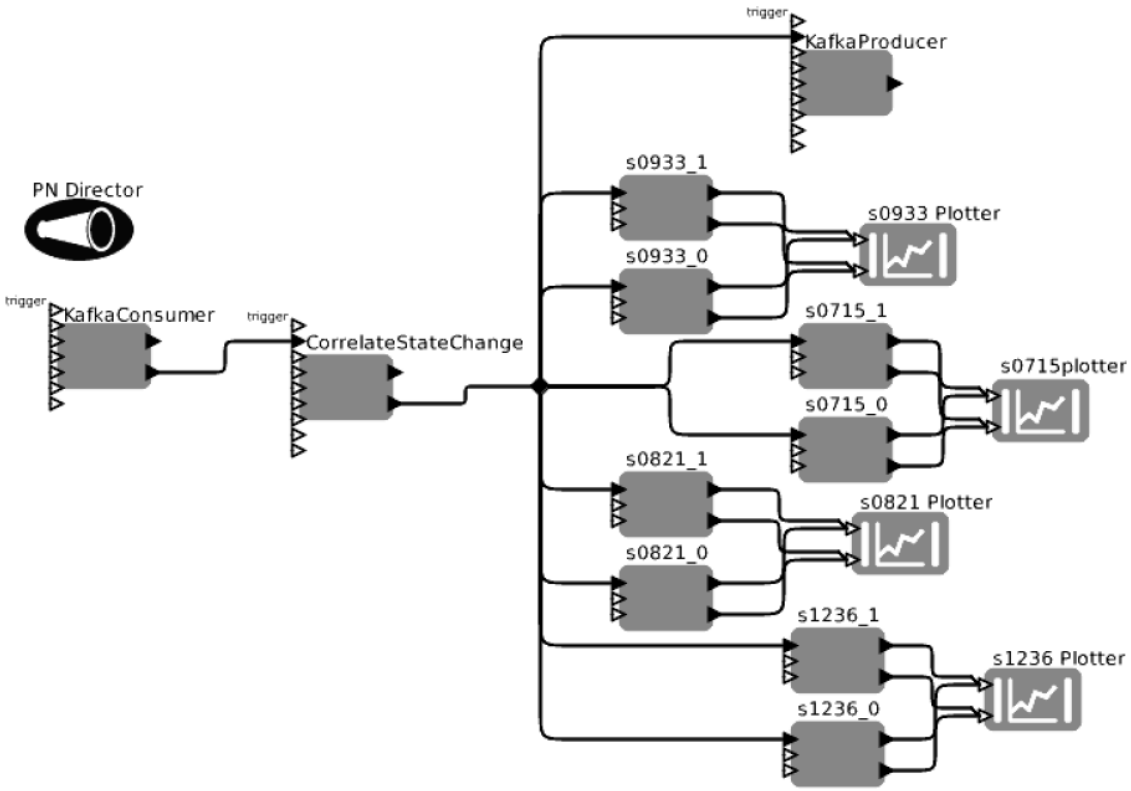

Figure 6. MWF_2 in the developed Kepler model

starts with new parameters automatically without GUI access. Figure 5 schematically shows the two options for organizing the launch of the micro-workflow described above.

In the current experiments, we have developed a Docker image that contains micro-workflow implementations using Kepler (Kepler MWF). To run a container, we need to pass the environment variables into the process. These variables include:

- MWF: the name of MWF;
- KAFKASERVER: the location of the Kafka server;
- KAFKAPORT: the port of the Kafka server endpoint;
- SCHEMAREGISTRY: the location of schema registry server;
- SCHEMAPORT: the port of the schema registry server endpoint;
- TOPICSOURCE: Kafka source topic;
- TOPICDESTINATION: Kafka destination topic.

Figure 7 shows the docker run command schema used to run a micro-workflow container.

```
docker run -it -d -p $KAFKAPORT -p $SCHEMAPORT \
        -e MWF=<?> -e KAFKASERVER=<?> \
        -e SCHEMAREGISTRY=<?> -e TOPICSOURCE=<?> \
        -e TOPICDESTINATION=<?> <developed_docker_image>
```

Figure 7. Docker command schema used to run a micro-workflow container

3.2. Experiment Deployment

We have implemented two deployment layouts for our micro-workflows during the experiment: a local deployment on a single node and a deployment in a distributed environment, where each micro-workflow has been deployed on a separate node.

The first deployment (see Fig. 8) on a single node acts as a benchmark for evaluating system performance, excluding delays introduced by network equipment and data exchange over the network. The streaming middleware, sensor emulator, MWF_1, and MWF_2 each packed in separate containers, deployed on a single computing node (Intel Core i7-4600U 2.1 GHz dual-core processor with 8 GB of RAM).

The second deployment is implemented on the resources of the Tornado Supercomputer at South Ural State University (Fig. 9). The Sensor Emulator virtual machine ($VM1$) simulates the process of IoT sensor data generation. It consumes data from DEBS 2012 database and publishes it sequentially into the Kafka input topic, deployed in the Landoop container ($VM2$). We partition the original DEBS 2012 first query workflow into two Micro-Workflows, MWF_1 and MWF_2, and pack each Micro-Workflow in a separate container, deployed on $VM2$ and $VM3$ virtual machines. There is no direct connection between the Virtual machines and nodes. All the communications need to cross a network server. Both $VM1$ and $VM3$ run on the same physical node (4 GB RAM and 4 cores of Intel Xeon X5680 CPU). $VM2$ runs on a separate physical node (12 GB RAM and 8 cores of Intel Xeon X5680 CPU). The interconnection between virtual machines is organized via an external physical node that acts as a virtual router, connected via a Gigabit Ethernet network.

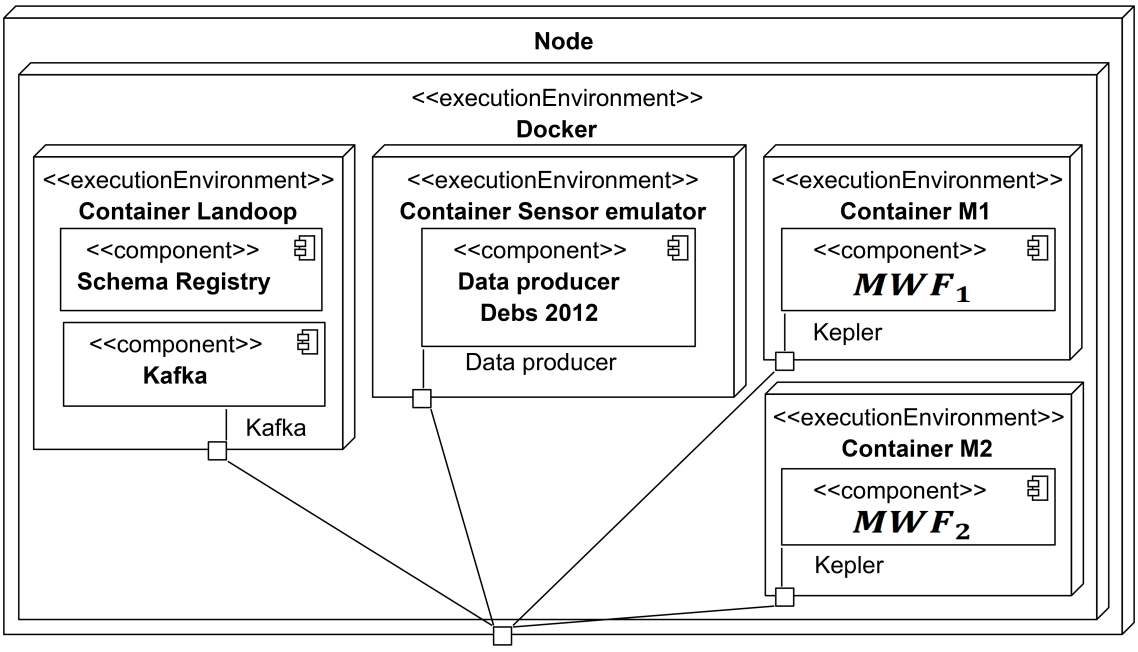

Figure 8. Local deployment on a single node

3.3. Evaluation

The following evaluation criteria are proposed to estimate the feasibility and overhead of the micro-workflow refactoring:

- AV_{sm}: the average interval between sensor messages.

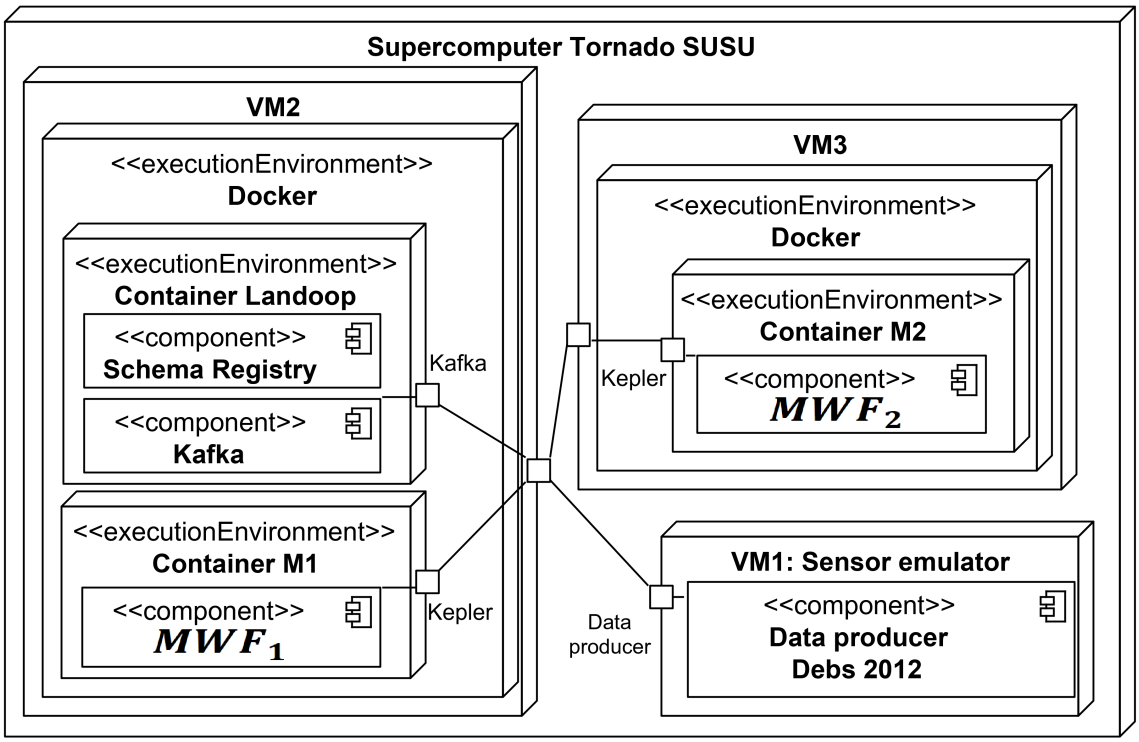

Figure 9. Deployment in a distributed computing environment on three nodes

- AV_{tat} (average processing time): the average time interval required to create one final message by a micro-workflow. The interval counted from the second requested original message has been received from the original Kafka topic to the time the final message sent to Kafka.
- AV_{dv} (average delivery time): average time interval between sending sensor message and receiving the message in MWF, including data serialization, message transfer from VM1 to VM2.
- AV_{l12}: the average network latency between VM1 and VM2.

The comparison results of the experiments are presented in Tab. 1.

Table 1. The comparison results

Parameters	Single node	Distributed mode
Testing time	24 hours	24 hours
Total messages	9 180 056	7 328 844
AV_{sm}	9.5 ms	11.8 ms
AV_{tat}	1.3 ms	3.2 ms
AV_{dv}	1.2 ms	4.4 ms
AV_{l12}	0 ms	3.5 ms

Analysis of the results of the experiment allows us to draw the following conclusions:

- Application of the workflow partitioning mechanism into independent micro-workflows allows providing IoT data processing in streaming mode, close to real-time. The average processing time in both experiments has been significantly less than the sensor's period of initial data generation.

- Computational processes of individual micro-workflows can be distributed across nodes in a distributed computing network. With such distribution, the data processing time increases by at least the latency between the computing network's corresponding nodes and the node where the event streaming platform is located.

Conclusion

We propose a model of workflow applications for stream processing in highly distributed environments such as fog computing. The ability to move part of computing from/into different nodes at a different level is the required by system architecture of fog computing. We show how tightly coupled dependencies in monolith workflow can be decoupled and refactored in the form of smaller and standalone micro-workflows and how define inputs and outputs of each resulting micro-workflow. Such decoupling supports the independence of implementation, execution, development, maintenance, and cross-platform deployment of micro-workflows as standalone computational units. An important direction of further research is to automate the refactoring workflow process into micro-workflows.

Acknowledgements

The reported study was funded by the Ministry of Science and Higher Education of the Russian Federation (government order FENU-2020-0022) and by RFBR, project number 19-37-90073.

References

1. Aazam, M., Zeadally, S., Harras, K.A.: Deploying Fog Computing in Industrial Internet of Things and Industry 4.0. IEEE Transactions on Industrial Informatics 14(10), 4674–4682 (2018), DOI: 10.1109/TII.2018.2855198

2. Alaasam, A.B.A., Radchenko, G., Tchernykh, A., Borodulin, K., Podkorytov, A.: Scientific Micro-Workflows: Where Event-Driven Approach Meets Workflows to Support Digital Twins. In: Proceedings of the International Conference Russian Supercomputing Days (RuSCDays'18), 24-25 Sept. 2018, Moscow, Russia. vol. 1, pp. 489–495 (2018)

3. Alaasam, A.B.A., Radchenko, G., Tchernykh, A.: Stateful Stream Processing for Digital Twins: Microservice-Based Kafka Stream DSL. In: 2019 International Multi-Conference on Engineering, Computer and Information Sciences (SIBIRCON), 21-27 Oct. 2019, Novosibirsk, Russia. pp. 0804–0809. IEEE (2019), DOI: 10.1109/SIBIRCON48586.2019.8958367

4. Alaasam, A.B.A., Radchenko, G., Tchernykh, A., González Compeán, J.L.: Analytic Study of Containerizing Stateful Stream Processing as Microservice to Support Digital Twins in Fog Computing. Programming and Computer Software 46(8), 511–525 (2020), DOI: 10.1134/S0361768820080083

5. Badia, R.M., Ayguade, E., Labarta, J.: Workflows for science: a challenge when facing the convergence of HPC and Big Data. Supercomputing Frontiers and Innovations 4(1), 27–47 (2017), DOI: 10.14529/jsfi170102

6. Bonomi, F., Milito, R., Zhu, J., Addepalli, S.: Fog computing and its role in the internet of things. In: Proceedings of the First Edition of the MCC Workshop on Mobile Cloud Computing, 13-17 Aug. 2012, Helsinki, Finland. pp. 13–15. Association for Computing Machinery, New York, NY, USA (2012), DOI: 10.1145/2342509.2342513

7. Boyes, H., Hallaq, B., Cunningham, J., Watson, T.: The industrial internet of things (IIoT): An analysis framework. Computers in Industry 101(December 2017), 1–12 (2018), DOI: 10.1016/j.compind.2018.04.015

8. Cao, J., Zhang, Q., Shi, W.: Challenges and Opportunities in Edge Computing, pp. 59–70. Springer, Cham (2018), DOI: 10.1007/978-3-030-02083-5_5

9. Carvalho, O., Roloff, E., Navaux, P.O.: A Distributed Stream Processing based Architecture for IoT Smart Grids Monitoring. In: Companion Proceedings of the 10th International Conference on Utility and Cloud Computing, 5-8 Dec. 2017, Austin, Texas, USA. pp. 9–14. ACM, New York, NY, USA (2017), DOI: 10.1145/3147234.3148105

10. Chandy, K.M.: Event Driven Architecture. In: Encyclopedia of Database Systems, pp. 1040–1044. Springer US, Boston, MA (2009), DOI: 10.1007/978-0-387-39940-9_570

11. Deelman, E., Singh, G., Su, M.H., et al.: Pegasus: A framework for mapping complex scientific workflows onto distributed systems. Scientific Programming 13(3), 219–237 (2005), DOI: 10.1155/2005/128026

12. Goyal, P., Mikkilineni, R.: Policy-Based Event-Driven Services-Oriented Architecture for Cloud Services Operation & Management. In: 2009 IEEE International Conference on Cloud Computing, 21-25 Sept. 2009, Bangalore, India. pp. 135–138. IEEE (2009), DOI: 10.1109/CLOUD.2009.76

13. Haag, S., Anderl, R.: Digital twin – Proof of concept. Manufacturing Letters 15, 64–66 (2018), DOI: 10.1016/j.mfglet.2018.02.006

14. Hao, Z., Novak, E., Yi, S., Li, Q.: Challenges and Software Architecture for Fog Computing. IEEE Internet Computing 21(2), 44–53 (2017), DOI: 10.1109/MIC.2017.26

15. Hirales-Carbajal, A., Tchernykh, A., Roblitz, T., Yahyapour, R.: A Grid simulation framework to study advance scheduling strategies for complex workflow applications. In: 2010 IEEE International Symposium on Parallel & Distributed Processing, Workshops and Phd Forum (IPDPSW), 19-23 April 2010, Atlanta, GA, USA. pp. 1–8. IEEE (2010), DOI: 10.1109/IPDPSW.2010.5470918

16. Hirales-Carbajal, A., Tchernykh, A., Yahyapour, R., et al.: Multiple Workflow Scheduling Strategies with User Run Time Estimates on a Grid. Journal of Grid Computing 10(2), 325–346 (2012), DOI: 10.1007/s10723-012-9215-6

17. Iturriaga, S., Nesmachnow, S., Tchernykh, A., Dorronsoro, B.: Multiobjective Workflow Scheduling in a Federation of Heterogeneous Green-Powered Data Centers. In: 2016 16th

IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid), 16-19 May 2016, Cartagena, Colombia. pp. 596–599. IEEE (2016), DOI: 10.1109/CCGrid.2016.34

18. Kalske, M., Mäkitalo, N., Mikkonen, T.: Challenges When Moving from Monolith to Microservice Architecture. In: Current Trends in Web Engineering, 5-8 June 2017, Rome, Italy, pp. 32–47. Springer, Cham (2018), DOI: 10.1007/978-3-319-74433-9_3

19. Korambath, P., Wang, J., Kumar, A., Davis, J., Graybill, R., Schott, B., Baldea, M.: A Smart Manufacturing Use Case: Furnace Temperature Balancing in Steam Methane Reforming Process via Kepler Workflows. Procedia Computer Science 80, 680–689 (2016), DOI: 10.1016/j.procs.2016.05.357

20. Li, X., Song, J., Huang, B.: A scientific workflow management system architecture and its scheduling based on cloud service platform for manufacturing big data analytics. The International Journal of Advanced Manufacturing Technology 84(1-4), 119–131 (2016), DOI: 10.1007/s00170-015-7804-9

21. Luo, J., Yin, L., Hu, J., Wang, C., Liu, X., Fan, X., Luo, H.: Container-based fog computing architecture and energy-balancing scheduling algorithm for energy IoT. Future Generation Computer Systems 97, 50–60 (2019), DOI: 10.1016/j.future.2018.12.063

22. Miranda, V., Tchernykh, A., Kliazovich, D.: Dynamic Communication-Aware Scheduling with Uncertainty of Workflow Applications in Clouds. In: High Performance Computer Applications, 9-13 March 2015, Mexico City, Mexico, Communications in Computer and Information Science, vol. 595, pp. 169–187. Springer, Cham (2016), DOI: 10.1007/978-3-319-32243-8_12

23. Naseri, M., Towhidi, A.: Stateful Web Services: A Missing Point in Web Service Standards. In: Proceedings of the International MultiConference of Engineers and Computer Scientists 2007 (IMECS 2007). pp. 993–997. Hong Kong, China (2007)

24. Peiffer, C., L'Heureux, I.: System and method for maintaining statefulness during client-server interactions. (12) United States Patent (US8346848B2) (2013), `https://patents.google.com/patent/US8346848B2/en`

25. Plociennik, M., Zok, T., Altintas, I., et al.: Approaches to Distributed Execution of Scientific Workflows in Kepler. Fundamenta Informaticae 128(3), 281–302 (2013), DOI: 10.3233/FI-2013-947

26. Qamsane, Y., Chen, C.Y., Balta, E.C., et al.: A unified digital twin framework for real-time monitoring and evaluation of smart manufacturing systems. In: 2019 IEEE 15th International Conference on Automation Science and Engineering (CASE), 22-26 Aug. 2019, Vancouver, BC, Canada. pp. 1394–1401 (2019), DOI: 10.1109/COASE.2019.8843269

27. Radchenko, G., Alaasam, A.B., Tchernykh, A.: Micro-Workflows: Kafka and Kepler Fusion to Support Digital Twins of Industrial Processes. In: 2018 IEEE/ACM International Conference on Utility and Cloud Computing Companion (UCC Companion), 17-20 Dec. 2018, Zurich, Switzerland. pp. 83–88. IEEE (2018), DOI: 10.1109/UCC-Companion.2018.00039

28. Richards, M.: Software Architecture Patterns. O'Reilly Media, 1005 Gravenstein Highway North, Sebastopol, CA 95472 (2015)

29. Savchenko, D., Radchenko, G., Taipale, O.: Microservices validation: Mjolnirr platform case study. In: 2015 38th International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO), 25-29 May 2015, Opatija, Croatia. pp. 235–240. IEEE (2015), DOI: 10.1109/MIPRO.2015.7160271

30. da Silva, R.F., Pottier, L., Coleman, T., Deelman, E., Casanova, H.: WorkflowHub: Community Framework for Enabling Scientific Workflow Research and Development. In: 2020 IEEE/ACM Workflows in Support of Large-Scale Science (WORKS), 12 Nov. 2020, GA, USA. pp. 49–56. IEEE (2020), DOI: 10.1109/WORKS51914.2020.00012

31. Simpkin, C., Taylor, I., Harborne, D., Bent, G., Preece, A., Ganti, R.K.: Efficient orchestration of Node-RED IoT workflows using a Vector Symbolic Architecture. Future Generation Computer Systems 111, 117–131 (2020), DOI: 10.1016/j.future.2020.04.005

32. Tao, F., Sui, F., Liu, A., et al.: Digital twin-driven product design framework. International Journal of Production Research 57(12), 3935–3953 (2019), DOI: 10.1080/00207543.2018.1443229

33. Yang, P.C., Purawat, S., U. Ieong, P., et al.: A demonstration of modularity, reuse, reproducibility, portability and scalability for modeling and simulation of cardiac electrophysiology using Kepler Workflows. PLOS Computational Biology 15(3), e1006856 (2019), DOI: 10.1371/journal.pcbi.1006856

34. Zhang, Q., Cheng, L., Boutaba, R.: Cloud computing: state-of-the-art and research challenges. Journal of Internet Services and Applications 1(1), 7–18 (2010), DOI: 10.1007/s13174-010-0007-6

35. Zheng, C., Tovar, B., Thain, D.: Deploying high throughput scientific workflows on container schedulers with makeflow and mesos. Proceedings of the 17th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID 2017), 14-17 May 2017, Madrid, Spain pp. 130–139 (2017), DOI: 10.1109/CCGRID.2017.9