Big Data Pipeline Scheduling and Adaptation on the Computing Continuum

Dragi Kimovski, Christian Bauer, Narges Mehran and Radu Prodan
Institute of Information Technology
University of Klagenfurt
Klagenfurt, Austria
dragi.kimovski, christian.bauer, narges.mehran, radu.prodan@aau.at

2022 IEEE 46th Annual Computers, Software, and Applications Conference (COMPSAC) | 978-1-6654-8810-5/22/$31.00 ©2022 IEEE | DOI: 10.1109/COMPSAC54236.2022.00181

***Abstract*—The Computing Continuum, covering Cloud, Fog, and Edge systems, promises to provide on-demand resource-as-a-service for Internet applications with diverse requirements, ranging from extremely low latency to high-performance processing. However, eminent challenges in automating the resources management of Big Data pipelines across the Computing Continuum remain. The resource management and adaptation for Big Data pipelines across the Computing Continuum require significant research effort, as the current data processing pipelines are dynamic. In contrast, traditional resource management strategies are static, leading to inefficient pipeline scheduling and overly complex process deployment.**

To address these needs, we propose in this work a scheduling and adaptation approach implemented as a software tool to lower the technological barriers to the management of Big Data pipelines over the Computing Continuum. The approach separates the static scheduling from the run-time execution, empowering domain experts with little infrastructure and software knowledge to take an active part in the Big Data pipeline adaptation. We conduct a feasibility study using a digital healthcare use case to validate our approach. We illustrate concrete scenarios supported by demonstrating how the scheduling and adaptation tool and its implementation automate the management of the lifecycle of a remote patient monitoring, treatment, and care pipeline.

***Index Terms*—Scheduling, Adaptation, Computing Continuum, Fog and Edge computing, Resources management**

I. Introduction and motivation

Cloud computing is a disruptive paradigm that facilitates elastic on-demand resource-as-a-service provisioning for various Internet applications. The concurrent Internet applications encompass complex Big Data pipelines, which have a vast set of conflicting requirements, such as low execution time and communication latency. The requirements of the Big Data pipelines demand infrastructure design that pushes the services, traditionally bounded within centralized Cloud data centers, closer to their distributed data sources. The so-called *Computing Continuum*, which federates the Cloud services with emerging Edge and Fog resources, processes data closer to their sources, promising to reduce the network transfer latency and communication bottlenecks. However, eminent challenges in automating the management of Big Data pipelines across the Computing Continuum remain, including their effective scheduling and adaptation over heterogeneous resources from different providers [1], [2].

Overall, the resource management and adaptation for Big Data pipelines across the continuum requires significant research effort, as the current data processing pipelines are dynamic. In contrast, traditional resource management strategies are static, leading to inefficient pipeline scheduling and overly complicated process deployment. Besides, with the advent of microservices architectures and containerization, the scheduling methods have to consider the proper data processing characteristics, facilitate networking connectivity, and provide optimized configuration of the Big Data pipeline workflows with guaranteed scalability from the execution performance perspective.

To address these needs, we propose a *Scheduling and Adaptation Approach* (SAA), implemented as a *software tool* to lower the technological barriers to the management of Big Data pipelines over the Computing Continuum. SAA separates the static scheduling from the run-time execution, empowering domain experts with little infrastructure and software knowledge to take an active part in the Big Data pipeline adaptation. SAA covers three iterative phases that automate the scheduling, deployment and execution of Big Data pipelines in the Computing Continuum: 1) *Scheduling* of Big Data pipelines to support their execution over heterogeneous resources; 2) *Deployment* and execution monitoring of Big Data pipelines; 3) *Execution adaptation* to improve performance when fluctuations and faults appear.

Therefore, the main contributions of the paper include: 1) *Requirements identification* for scheduling and adapting Big Data pipelines on the Computing Continuum; 2) *Approach description and architecture design* consisting of two interoperable tools for scheduling and adaptation that jointly automate the Big Data pipelines deployment lifecycle; 3) *Feasibility study* using in a pilot case from the digital healthcare domain, targeting large-scale support for remote patient monitoring, treatment, and care.

The paper has eight sections and is organized as follows. Section II discusses the related work. Section III introduces the digital healthcare pilot use case for this paper and the requirements analysis. Section IV outlines the SAA scheduling and adaptation algorithms. Section V presents the data flow between the scheduling and adaptation algorithms. Section VI illustrates the technical architecture of the SAA tool, while Section VII provides a feasibility study using the healthcare

978-1-6654-8810-5/22/$31.00 ©2022 IEEE
DOI 10.1109/COMPSAC54236.2022.00181

pilot. Finally, Section VIII concludes the paper.

II. Related work

This section reviews the state-of-the-art in Big Data pipelines scheduling and adaptation across the Computing Continuum.

1) Traffic-aware scheduling and adaptation: Arkian et al. [3] presented a geo-distributed stream processing autoscaling model with the aim of maintaining a sufficient maximum sustainable throughput between edge devices. Their model optimizes the throughput per task of the Big Data pipeline. Tamiru et al. [4] designed an integrated scheduler in Kubernetes orchestration platform for multi-cluster computing environments. The authors modeled the resource requirement of the workload, the ingress traffic, and the resource capacity of each cluster to decide on an appropriate application deployment.

2) Cost-aware scheduling and adaptation: De Maio and Kimovski [5] investigated the potential of Fog computing for scheduling and adaptation of extreme data scientific workflows with the goal of optimizing processing time, reliability and user's monetary cost. Sharghivand et al. [6] proposed a two-sided matching solution to schedule Big Data analytics tasks on cloudlets as part of the Edge environment. Their model determines the costs of the Edge services based on cloudlet's and user's preferences to match the resources to applications.

3) Latency-aware scheduling and adaptation: Menouer [7] designed a multi-criteria decision-analysis model to solve the scheduling and adaptation problem, which considers criteria such as the utilization of resources, alongside the processing time the workflow tasks and data transmission time. Fahs and Pierre [8] proposed a latency-aware scheduler for a Fog computing infrastructure. The authors aimed to identify an appropriate replica scheduling model that minimizes the tail user-to-replica latency and balances the pipelines. Hoseiny et al. [9] designed a scheduling algorithm, based on an optimization model that is a weighted sum of overall pipeline processing time, energy consumption, and percentage of deadline satisfied tasks.

III. Requirements analysis

We present the methodology for designing the SAA scheduling and adaptation approaches, following a requirements analysis over a real life pilot use case.

A. Pilot use case: Digital healthcare

The digital healthcare use case provides solutions to medical institutions for improvement of patient diagnosis and treatment. Typical scenarios involve treatment and care of elderly, chronic disease (e.g., diabetes, kidney, COPD, cancer), or quarantined (e.g., COVID-19) patients. Examples of digital healthcare services offered to these patients include personal safety alarms, care phones, healthcare call centers, remote patient monitoring, and digital supervision.

Digital healthcare services control two types of sensors gathering raw information from thousands of patients through their tablets, mobile phones, sensors, or other devices: 1) *Remote supervision sensors* such as patient motion sensors and video cameras; 2) *Care and wellness-specific sensors* monitoring specific patient data, such as blood pressure, scales, glucose, and medicine quantity.

Gathering data from thousands of patients through these sensors requires complex Big Data pipelines that perform automated processing, including filtering, encryption, anonymization, and storage. Such pipelines may take different dynamic workflow structures depending on their functionality, patient type (e.g., elderly, quarantined), disease, data acquisition device, data format (e.g., alarm, measurement, video stream), stakeholder (e.g., doctor, nurse, response center), or third-party integration (e.g., electronic health records). Typical digital health pipeline examples comprise: 1) Medical devices regularly sending patient data to doctors or nurses for analysis and review; 2) Camera sensors sending critical alarms (e.g., leaving bed or room) to response centers, caregivers, physicians, or family members.

B. Requirements methodology

We defined the functionality of SAA through an iterative methodology based best practices, presented in [10] following the pilot use case described above.

We collected the requirements using two types of interviews: 1) *General interviews* to understand the working practices and data management needs; 2) *Dedicated ad-hoc interviews* with the company that developed the use case to understand the relevant requirements and functionalities.

C. Extracted requirements

The requirements for the Big Data pipelines scheduling and adaptation are extracted from two main sources: analysis of the state-of-the-art literature (presented in Section II) and the interviews with the pilot case provider. We therefore utilize the following requirements to guide the modeling process of SAA:

- Requirement-1: Metric Specific Application Scaling;
- Requirement-2: Limited Dynamic Scheduling;
- Requirement-3: Requirement Definition per Task;
- Requirement-4: QoS Guarantee for Tasks with Strict Deadlines;
- Requirement-5: Avoidance of Highly Utilized Resources;
- Requirement-6: Task Offloading on Edge and Mobile Devices;
- Requirement-7: Data-Aware Pipeline Scheduling;
- Requirement-8: Low Scheduling Overhead;
- Requirement-9: Schedule the tasks in transparent manner;
- Requirement-10: Identify resources on the run that can support high stream data rates;
- Requirement-11: Provide support for adaptation and offline scheduling.

Therefore, from the list of the requirements, we can conclude that the SAA scheduler and adaptation should provide means for low-latency scheduling, capable of offloading the task execution on Fog and Edge devices with avoidance of highly

utilized resources. Besides, SAA should be able to identify resources that can support processing of high data streams, while maintaining the latency for processing the requests of the use case applications.

D. Challenges

We extracted several challenges from the requirement analysis presented above in Section III-B.

1) Pipeline scheduling: challenges such as: 1) Increased resource utilization and application performance; 2) Task offloading over various control domains across the Computing Continuum; 3) Scalable pipeline management and elastic resource provisioning upon peak patient traffic (e.g., epidemics, pandemics) for a highly responsive operation.

2) Pipeline adaptation: faces challenges such as: 1) real-time monitoring and notification upon exceptional events; 2) identification of pipeline execution anomalies and defining of proper mitigation rules; 3) adaptation of the resources when the mitigation rules are not able to correct the execution anomalies.

IV. SAA SCHEDULING AND ADAPTATION

We present in this section the scheduling and adaptation algorithms together with the conceptual execution workflow.

A. Scheduling

We represent a *scheduling problem* as a matching game using two disjoint sets of players [11]:

1) a set of tasks $\mathcal{M}$ of the Big Data pipeline workflow $\mathcal{W}$, and
2) a set of resources $\mathcal{R}$.

The game aims to match a task $m_i \in \mathcal{M}$ to a resource $r_j \in \mathcal{R}$ with sufficient processing capacity that ranks the players and optimizes the aggregated utility of the Big Data pipeline and the Computing Continuum resources. Therefore, our aim is to maximize the aggregated utility function by determining based on specific tasks and resources ranking. We, therefore, define ranking strategies and the aggregate utility functions as:

a) Task-side ranking: orders the resources for a task m_i in a *resource preference list* $\texttt{RPL}[m_i]$ based on the aggregated pipeline processing and queuing times:

$$T_{pq}\left(m_i, \texttt{pip}_{ui}, r_j\right) = T_p\left(m_i, \texttt{pip}_{ui}, r_j\right) + T_q\left(m_i, \texttt{pip}_{ui}, r_j\right),$$

where $\texttt{pip}_{ui}$ is data exchange from upstage task m_u to downstage task m_i, $T_p\left(m_i, \texttt{pip}_{ui}, r_j\right)$ is pipeline processing time and $T_q\left(m_i, \texttt{pip}_{ui}, r_j\right)$ is pipeline queuing time.

b) Task-side utility: represents the gain obtained by a task m_i scheduled on one of its preferred resources $r_j \in \texttt{RPL}\left[m_i\right]$:

$$U_s\left(m_i, r_j\right) = \frac{\left|T_{pq}\left(m_i, \texttt{pip}_{ui}, r_j\right) - \textsc{Last}_{T_{pq}}(\texttt{RPL}[m_i])\right|}{\left|\textsc{First}_{T_{pq}}(\texttt{RPL}[m_i]) - \textsc{Last}_{T_{pq}}(\texttt{RPL}[m_i])\right|},$$

where $\mathrm{FIRST}_{T_{pq}}(\texttt{RPL}[m_i])$ and $\mathrm{LAST}_{T_{pq}}(\texttt{RPL}[m_i])$ lowest, respectively highest Big Data pipeline processing and queuing times of the resources in $\texttt{RPL}\left[m_i\right]$. The first resource in the preference list $\texttt{RPL}\left[m_i\right]$ provides the highest utility equals to one.

c) Resource-side ranking: ranks the tasks for a resource r_j in a *task preference list* $\texttt{TPL}[r_j]$ based on the pipeline transmission time $T_c\left(m_u, \texttt{pip}_{ui}, m_i\right)$.

d) Resource-side utility: presents the gain obtained by a resource r_j for executing one preferred task $m_i \in \texttt{TPL}\left[r_j\right]$:

$$U_r\left(m_i, r_j\right) = \frac{\left|T_c\left(m_u, \texttt{pip}_{ui}, m_i\right) - \textsc{Last}_{T_c}(\texttt{TPL}[r_j])\right|}{\left|\textsc{First}_{T_c}(\texttt{TPL}[r_j]) - \textsc{Last}_{T_c}(\texttt{TPL}[r_j])\right|},$$

where $\mathrm{FIRST}_{T_c}(\texttt{TPL}[r_j])$ and $\mathrm{LAST}_{T_c}(\texttt{TPL}[r_j])$ represent the pipeline transmission time of first, respectively last task in the preference list $\texttt{TPL}[r_j]$.

e) Scheduling algorithm: Algorithm 1 applies a level-based method to schedule independent tasks $\mathcal{D}(d)$ of a level d of Big Data data pipeline workflow $\mathcal{W}$, identified based on the dependencies between the tasks. The algorithm receives the pipeline workflow $\mathcal{W} = (\mathcal{M}, \mathcal{E}, \mathcal{Q}, \mathcal{M}_{\texttt{src}}, \mathcal{M}_{\texttt{snk}}, \mathcal{D})$ as an input, where $\mathcal{M}$ is the set of Big Data pipeline tasks, $\mathcal{E}$ is the Big Data pipelines, $\mathcal{Q}$ is the data queue between the tasks, $\mathcal{M}_{\texttt{src}}$ is the data producers (sources) and $\mathcal{M}_{\texttt{snk}}$ is the data consumers (sinks). Therefore, the algorithm searches for the best schedule $\texttt{sched}(m_i) \leftarrow \mu[d, m_i]$ of $\mathcal{W}$ on the available resources $\mathcal{R}$ and identifies the earliest start time $\texttt{EST}\left(m_i\right)$ of each task.

After initializing the resource preference list and matching lists, (lines 2–3), the algorithm loops over the dependency levels of the Big Data pipeline to rank the resources for independent tasks (line 6), and to rank the tasks for the resources (lines 7–13). After ranking, the algorithm attempts to find the appropriate matches based on the available resources (line 14). The matching function matches the task to one of its preferred resources, and then allocates the resource due to task's required capacity. The matching function loops over the task and resource preference lists, in addition to checking for the over capacity and the full capacity of resources. Therefore, the matching reaches an equilibrium while no task or resource prefers other choices than their current matched choices. Afterward, the scheduling algorithm estimates the earliest start time and assigns the appropriate resource to the task m_i (lines 15–18). Finally, the algorithm returns the earliest start time and scheduling plan for the Big Data pipeline tasks (line 20).

B. Adaptation

The SAA adaptation approach is defined as a loop, that cyclically updates every component contained in the loop. The adaptation loop is shown in figure 1 and consists of the Big Data pipeline and the *resources* registered and monitored to the *Monitoring and Analysis* component. Both the Big Data pipeline and the resources send the gathered monitoring information as a state at a time step P_t or S_t respectively to the *Monitoring and Analysis* component, where t denotes the current time step.

The Monitoring and Analysis component then uses thestates P_t and S_t, and calculates an action A_{t+1} as an update for the resources, where $t + 1$ denotes the next time step.

Algorithm 1 Scheduling algorithm.

Input: $\mathcal{W} = (\mathcal{M}, \mathcal{E}, \mathcal{Q}, \mathcal{M}_{\text{src}}, \mathcal{M}_{\text{snk}}, \mathcal{D})$ ▷ Big data pipeline workflow
$\mathcal{R} = \{r_j \mid 0 \leq j < \mathcal{N}_{\mathcal{R}}\}$ ▷ Cloud, Fog, and Edge resource set
$\mathcal{L} = \{l_{kj} \mid 0 \leq k, j < \mathcal{N}_{\mathcal{R}}\}$ ▷ Channel set
Output: $\texttt{EST}(m_i), \texttt{sched}(m_i); \forall m_i \in \mathcal{W}$ ▷ Start time and scheduling of all tasks

function SCHEDULING($\mathcal{W}, \mathcal{R}, \mathcal{L}$)
  $Initialize(\texttt{sched})$ ▷ Initialize scheduling list
  $Initialize(\mu)$ ▷ Initialize matching lists
  $d \leftarrow 1$
  for all $d \leq$ SIZE(D) **do**
    RPL $\leftarrow$ RANKRESOURCES($\mathcal{W}, \mathcal{R}, \mathcal{L}, d$) ▷ Rank resources for tasks of level d
    if $d = 1$ **then** ▷ Rank tasks of level 1 for resources
      TPL $\leftarrow$ RANKTASKS($\mathcal{W}, \mathcal{R}, \mathcal{L}, d$, RPL, $\emptyset$)
    else
      if $d \neq 1$ **then** ▷ Rank tasks of level d for resources
        TPL $\leftarrow$ RANKTASKS($\mathcal{W}, \mathcal{R}, \mathcal{L}, d$, RPL, $\mu[d-1, ^*]$)
      end if
    end if
    $\mu[d, ^*] \leftarrow$ MATCH($\mathcal{W}, \mathcal{R}, d$, TPL, RPL) ▷ Match independent tasks to resources
    for all $m_i \in \mathcal{D}(d) \ \wedge \ (m_u, m_i, \texttt{pip}_{ui}) \in \mathcal{E} \ \wedge \ \texttt{qu}_{ui} \in \mathcal{Q}$ **do**
      $\texttt{EST}(m_i) \leftarrow \min_{\forall (m_u, m_i, \texttt{pip}_{ui}) \in \mathcal{E} \wedge \texttt{qu}_{ui} \in \mathcal{Q}} \left(\texttt{EST}(m_u) + T_{pq} + T_c \right)$
      $\texttt{sched}(m_i) \leftarrow \mu[d, m_i]$
    end for
  end for
  return (EST, sched);
end function

This received action A_{t+1} is then used by the Resource components to reconfigure the resources if necessary. The adaptation uses monitoring of tasks and resources to retrieve the states P_t, S_t.

Monitoring component is necessary to obtain information about failures or performance fluctuations along with under-, over-utilization of resources. The state, therefore, provides feedback if a resource is able to handle additional load, and thus, more tasks will potentially be mapped to it for execution. In case that the resource is not capable of handling the current load, less demanding tasks will be mapped to that resource in future scheduling cycles and it will be reconfigured accordingly. A monitored resource information consists of the CPU, memory and storage utilization, in addition to network bandwidth usage. The Big Data pipeline and resources send their states P_t, S_t respectively to the Monitoring and Analysis component. Afterward, given those states, the analyzer will determine the next action A_{t+1} of the resources. The monitoring feedback will be periodically retrieved from all registered resources and provided to the analyzer. In case of a resource side failure, the monitoring mechanism will be alerted of the occurred anomaly. The analyzer uses this monitoring data to decide if any actions $a \in A$ on the current scheduling plan have to be applied.

The following actions A are considered in the adaptation approach:

Resource load within expected parameters: When tasks on resources are well mapped and no action has to be taken for these resources, this is denoted in the action set A_{t+1} as such. This state is achieved when a resource is running within expected parameters, the estimated time of completing a task is not exceeded or other anomalies occur.

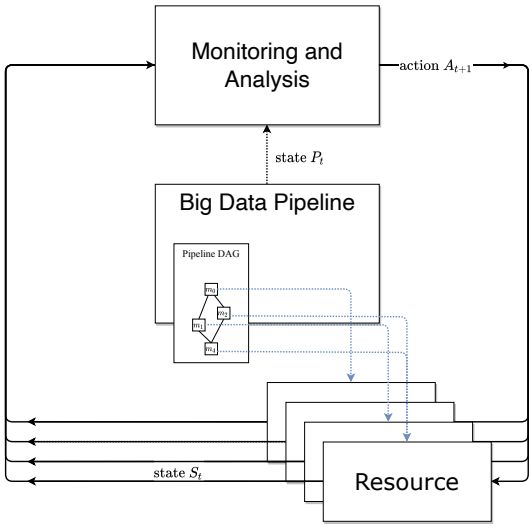

Fig. 1. Adaptation loop.

Resource load under expected parameters: When the monitoring component detects under-utilization of a resource, tasks of the pipeline are mapped to it to be efficiently utilized.

Resource load over expected parameters: If a resource is over-utilized, there are multiple options. If it is detected that horizontal scaling solves the over-utilization and the resource instance is capable of being scaled up, it will be reconfigured to be within expected utilization levels. Furthermore, if an ill-defined task is detected and was mapped to a resource that isn't capable to fulfill the computation within a reasonable time frame, this task will be migrated to another resource.

V. SAA SCHEDULING AND ADAPTATION INTEGRATION WORKFLOW

In this section, we provide detailed description of the interactions between the scheduling and adaptation algorithms with the possible deployment frameworks.

Step-1: To begin with, the Big Data application owner submits the pipeline for deployment over the Computing Continuum. In addition the owner provides information on the pipeline structure, tasks, and executable data.

Step-2: After receiving the structure of the pipeline, the scheduling algorithm performs dependency analysis between the tasks in $\mathcal{D}(d)$ of a level d to identify the correct sequence of tasks and those that can be executed in parallel.

Step-3: Afterwards, the scheduling algorithm analyses the requirements of each specific task, such as processing speed, memory, and storage sizes.

Step-4: Based on the identified task requirements, the scheduling algorithm searches for appropriate resources with associated performance metadata.

Step-5: The scheduling algorithm analyzes the resources based on the requirements of the pipeline tasks, creates a schedule $\texttt{sched}(m_i) \leftarrow \mu[d, m_i]$ for every pipelines task, and sends it to a deployment and run-time system (such as Kubernetes [12]).

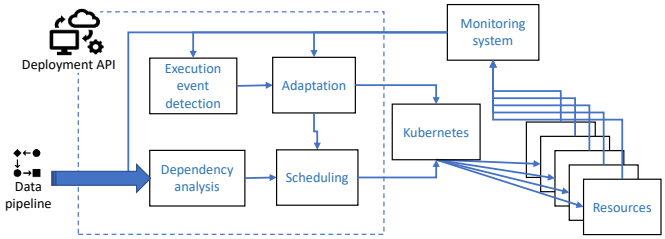

Fig. 2. Scheduling and adaptation tool architecture.

Step-6: The deployment system deploys the pipeline tasks on the resources following the schedule provided by the scheduling algorithm.

Step-7: A monitoring system observes the execution of the Big Data pipeline and the behavior of the computing continuum resources.

Step-8: The monitoring information is sent to the adaptation algorithm for analysis of service level objective (SLO) violations and over-utilization (such as high `CPU` load or low amount of operating memory available).

Step-9: If over-utilization is identified, the adaptation algorithm uses rule-based mitigation strategy. For example, in the cases where there is no available operating memory, a migration of the Big Data pipeline will be performed.

Step-10: The adaptation algorithm continuously monitors the execution of the Big Data pipeline, and applies the rule-based mitigation actions A until the pipeline finishes with its execution.

VI. Scheduling and Adaptation Functional Architecture

We present in this section the architecture of SAA in one integrated tool in detail.

The architecture of the integrated SAA scheduling and adaptation tool consists of five components, displayed in Figure 2.

a) Dependency analysis: orders the candidate tasks scheduled on each resource in a preference list based on the aggregate pipeline communication time to each task.

b) Execution event detection: uses data from external monitoring system to identify SLO violations or anomalies during execution.

c) Scheduling: maps the pipeline tasks to the resources using a matching theory algorithm, applied on the task and resource preference lists in response to infrastructure drifts.

d) Adaptation: dynamical applies re-scheduling or migration of the tasks based on the analyzed monitoring received from the execution event detection. Both the scheduling and adaptation interact with deployment and orchestration system to perform deployment and adaptation of the tasks.

e) Public API: enables the integration of the SAA scheduling and adaptation tool with deployment and orchestration systems.

VII. Feasibility Study: Patient Monitoring Pipeline

We present a feasibility study and the benefit of applying the SAA scheduling and adaptation integrated tool to automate pipeline execution. The pipeline offers in-house patient care through vital sign monitoring and real-time analysis, identifying critical medical conditions. The pipeline gathers continuously multi-modal medical data from various on-body medical sensors, including pulse, blood pressure, and blood saturation readings. Afterward, it aggregates, cleans, and anonymizes the data close to the sensors and stores it in a secure database. The pipeline continuously analyses the stored data for signs of critical medical conditions and provides it to practitioners. The use case pipeline is described in details in Section III.A.

A. Implementation

We integrate the SAA scheduling and adaptation tool as a default scheduler within the Kubernetes (version 1.2) open-source container-orchestration system for automating computer application deployment, scaling, and management. We implement SAA in Python version 3.9, using the `daffidwilde` [13] matching library, and the resource-side and task-side rankings using a non-dominated sorting algorithm. We deploy it as a RESTful service in the *Carinthian Computing Continuum (C^3)* infrastructure testbed, providing a rich set of resources across the Cloud, Fog, and Edge layers, illustrated in Table I.

B. Patient monitoring pipeline scheduling and adaptation

SAA receives the digital health pipeline from the application owner and arranges for its adaptive execution in two phases.

a) Pipeline analysis: investigates the requirements and dependencies of the pipeline tasks, as they may have different execution orders and priorities. Besides, it considers the network distance to the patients' medical sensors to improve locality and reduce the latency for the time-critical communication between the data sources and the personal health gateway. The pipeline received from the application owner consists of Docker container images, which allow task execution across various Edge or Cloud resources. Afterward, SAA creates a configuration file in YAML format. The file contains information about the image of the task, tasks requirements, preferred resources, operating system, and processor architecture. Concretely, it defines the three task instances instantiated and connected with the `RetrieveSaturation` containers gathering medical sensor data. Furthermore, it provides information from where to acquire the `retrievemedicaldata` image and on which device to execute (e.g., Raspberry Pi4, `datacloud/ubuntu-retrievemedicaldata:rpi4` image).

b) Pipeline scheduling and adaptation: creates a schedule for the pipeline using a matching theory algorithm and provides it to Kubernetes for deployment in JSON format. Table II offers a sample schedule of

TABLE I
CARINTHIAN COMPUTING CONTINUUM (C^3) TESTBED [14].

Infrastructure layer	*Device / Instance type*	*Architecture*	*(v)CPU*	*Memory [GiB]*	*Storage [GiB]*	*Network*	*Physical processor*	*Clock [GHz]*	*Operating system*
Cloud layer	`AWS t2.micro`	64-bit x86	1	1	32	Moderate	Intel Xeon	≤ 3.1	Ubuntu 18.04
	`AWS c5.large`		2	4		≤ 10 Gbit s^{-1}	Intel Xeon Platinum 8000 series	≤ 3.6	
	`AWS m5a.xlarge`		4	16			AMD EPYC 7000 series	≤ 2.5	
Fog layer	`Exoscale Tiny`	64-bit x86	1	1	32	≤ 10 Gbit s^{-1}	Intel Xeon	≤ 3.6	Ubuntu 18.04
	`Exoscale Medium`		2	4					
	`Exoscale Large`		4	8					
	`ITEC Cloud Instance`		4	8			Intel Xeon Platinum 8000	≤ 3.1	
Edge layer	`Edge Gateway System`	64-bit x86	12	32	32	≤ 10 Gbit s^{-1}	AMD Ryzen Threadripper 2920X	≤ 3.5	Ubuntu 18.04
	`Raspberry Pi 3B`	64-bit ARM	4	1	64	≤ 1 Gbit s^{-1}	Cortex - A53	≤ 1.4	Pi OS Buster
	`Raspberry Pi 4`			4			Cortex - A72	≤ 1.5	
	`Jetson Nano`			4			Tegra X1 and Cortex - A57	≤ 1.43	Linux for Tegra R28.2.1

TABLE II
PATIENT MONITORING PIPELINE SCHEDULE.

No.	*Task*	*Resource*	*Task dependency*
1	`RetreivePulseReading 1`	Raspberry Pi 4	/
2	`RetrieveBloodPressure 1`	Raspberry Pi 4	/
3	`RetrieveSaturation 1`	Raspberry Pi 4	/
4	`SaveRetrievedPatientData`	Edge Gateway System	1, 2, 3
5	`EdgeProcessingAnonymization`	Edge Gateway System	3
6	`EdgeProcessingCleaningData`	Jetson Nano	3
7	`CloudProcessingAllPatientData`	AWS `m5a.xlarge`	5, 6

the pipeline over the C^3 resources. 1) *Raspberry Pi 4* devices, despite lower performance, are appropriate for aggregating sensor data in real-time, as required by the `RetreivePulseReading`, `RetrieveBloodPressure` and `RetrieveSaturation` tasks; 2) *Edge Gateway System* has higher performance to execute processing and storage demanding tasks, such as `SaveRetrievedPatientData` and `EdgeProcessingAnonymization`. 3) *Jetson Nano* devices have specialized accelerators that cleanup data in nearly real-time, as required by `EdgeProcessingCleaningData`. 4) *AWS instances* host the cross patient data analysis performed by the `CloudProcessingAllPatientData` task.

VIII. CONCLUSIONS

We presented a novel scheduling and adaptation approach for automating the lifecycle of Big Data pipelines execution in the Computing Continuum. The SAA approach separates the scheduling of the Big Data pipelines from the run-time adaptation stage, empowering domain experts with little infrastructure and resources know-how to participate in their operation actively. SAA introduced a two-sided matching-based scheduling method integrated within the C^3 infrastructure that considers the Big Data pipeline processing and transmission times. In addition, SAA is capable to adapt the execution of the pipeline on-the-fly by employment of state model to improve the execution the tasks based on their run-time requirements.

We, therefore, presented the detailed architectural design of the SAA tool and conducted a feasibility study using a digital healthcare pilot case targeting remote patient monitoring, treatment, and care. We presented concrete scenarios supported by authentic snapshots demonstrating how the SAA implementation automate the management of the lifecycle of a digital healthcare data pipeline on a real Computing Continuum infrastructure.

ACKNOWLEDGEMENT

This work is supported by the DataCloud Project funded by the Horizon 2020 Programme.

REFERENCES

[1] 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 Comput. Surv.*, vol. 52, no. 5, Sep. 2019. [Online]. Available: https://doi.org/10.1145/3332301

[2] A. R. Munappy, J. Bosch, and H. H. Olsson, "Data pipeline management in practice: Challenges and opportunities," in *Product-Focused Software Process Improvement*, M. Morisio, M. Torchiano, and A. Jedlitschka, Eds. Cham: Springer International Publishing, 2020, pp. 168–184.

[3] H. Arkian, G. Pierre, J. Tordsson, and E. Elmroth, "Model-based stream processing auto-scaling in geo-distributed environments," in *ICCCN 2021-30th International Conference on Computer Communications and Networks*, 2021.

[4] M. Tamiru, G. Pierre, J. Tordsson, and E. Elmroth, "mck8s: An orchestration platform for geo-distributed multi-cluster environments," in *ICCCN 2021-30th International Conference on Computer Communications and Networks*, 2021.

[5] V. De Maio and D. Kimovski, "Multi-objective scheduling of extreme data scientific workflows in fog," *Future Generation Computer Systems*, vol. 106, pp. 171 – 184, 2020. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0167739X19309197

[6] N. Sharghivand, F. Derakhshan, L. Mashayekhy, and L. Mohammad Khanli, "An edge computing matching framework with guaranteed quality of service," *IEEE Transactions on Cloud Computing*, pp. 1–1, 2020.

[7] T. Menouer, "Kcss: Kubernetes container scheduling strategy," *The Journal of Supercomputing*, vol. 77, no. 5, pp. 4267–4293, 2021.

[8] A. J. Fahs and G. Pierre, "Tail-latency-aware fog application replica placement," in *International Conference on Service-Oriented Computing*. Springer, 2020, pp. 508–524.

[9] F. Hoseiny, S. Azizi, M. Shojafar, F. Ahmadiazar, and R. Tafazolli, "Pga: A priority-aware genetic algorithm for task scheduling in heterogeneous fog-cloud computing," in *IEEE INFOCOM 2021 - IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, 2021, pp. 1–6.

[10] B. Nuseibeh and S. Easterbrook, "Requirements engineering: a roadmap," in *Proceedings of the Conference on the Future of Software Engineering*, 2000, pp. 35–46.

[11] D. Gale and L. S. Shapley, "College admissions and the stability of marriage," *The American Mathematical Monthly*, vol. 69, no. 1, pp. 9–15, 1962.

[12] M. Luksa, *Kubernetes in action*. Simon and Schuster, 2017.

[13] H. Wilde, V. Knight, and J. Gillard, "Matching: A python library for solving matching games," *Journal of Open Source Software*, vol. 5, no. 48, p. 2169, 2020.

[14] D. Kimovski, R. Mathá, J. Hammer, N. Mehran, H. Hellwagner, and R. Prodan, "Cloud, fog or edge: Where to compute?" *IEEE Internet Computing*, 2021.