

ICT-40-2020
H2020-ICT-2018-2020



DataCloud

ENABLING THE BIG DATA PIPELINE LIFECYCLE ON THE COMPUTING CONTINUUM

D5.1: DATA-AWARE RESOURCE PROVISIONING ACROSS THE COMPUTING CONTINUUM

Document Metadata

Work package	WP5
Date	17.12.2021
Deliverable editor	Dragi Kimovski (AAU) and Narges Mehran (AAU)
Version	1.0
Contributors	Radu Prodan (AAU)
Reviewers	Alexandre Ulisses (MOG) and Aleena Thomas (SINTEF)
Keywords	Scheduling, Matching theory, Execution optimization
Dissemination Level	Public

Document Revision History

Version	Date	Description of change	List of contributor(s)
V0.1	06.06.2021	1 st structure of the deliverable.	Dragi Kimovski (AAU)
V0.2	01.09.2021	1 st draft of the user manual.	Narges Mehran (AAU)
V0.3	01.09.2021	Model description.	Narges Mehran (AAU)
V0.4	01.10.2021	Related work.	Dragi Kimovski (AAU)
V0.5	01.10.2021	Trace example.	Dragi Kimovski (AAU)
V0.6	01.11.2021	Evaluation.	Narges Mehran (AAU)
V0.7	15.11.2021	Completion of 1 st draft.	Dragi Kimovski (AAU)
V0.8	01.12.2021	Reviewers' comments addressed.	Dragi Kimovski (AAU)
V1.0	17.12.2021	Final formatting and layout.	Brian Elvesæter (SI)

DISCLAIMER

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 101016835. This document reflects only the authors' views and the Commission is not responsible for any use that may be made of the information it contains.



EXECUTIVE SUMMARY

This document describes the research activities related to the resources provisioning and scheduling of Big Data pipelines and the related algorithms and tools developed within the framework of the DataCloud project. The document is primarily focused on the undertakings conducted in Task 5.1, namely the modelling and development of the ADA-PIPE tool. Therefore, first the requirements for the scheduling and provisioning algorithm have been analysed and, thus, a matching based scheduling algorithm was proposed. Sources for requirements discovering included state-of-the-art analysis using literature review and interviews carried with business case partners.

As a result of requirements analysis, we propose a tool and related algorithms for data-aware scheduling across the computing continuum. The main contributions of this work include:

- A model for quantifying the data pipeline processing and queuing time tailored to asynchronous data exchange by utilizing the data queues.
- A ranking strategy including the transmission time in the resource-side ranking to localize the processing of users' data and meet the response time quality of service.
- A modified two-sided stable matching model based on the resource utilization for scheduling data pipeline on resources.
- An algorithm for scheduling of complex data pipeline on resources based on a two-sided stable matching model by considering the resource utilization.

The proposed tool also supports integration with different orchestration systems (including MAESTRO and Kubernetes) and monitoring systems (including Kubernetes). The proposed tool has been validated on a real-life computing continuum infrastructure over a specifically tailored DataCloud use case application.



TABLE OF CONTENTS

1	INTRODUCTION	8
2	STATE OF THE ART	10
3	REQUIREMENTS ANALYSIS	11
4	SCHEDULING MODEL.....	12
5	SCHEDULING ALGORITHM.....	14
6	ADA-PIPE SCHEDULER TRACE EXAMPLE	16
7	PRELIMINARY EVALUATION	17
8	CONCLUSIONS	23
9	REFERENCES.....	24



LIST OF FIGURES

FIGURE 1: ADA-PIPE SCHEDULER TRACE EXAMPLE	16
FIGURE 2: TASK AND RESOURCES UTILITY FOR DIFFERENT NUMBER OF TASKS ..	21
FIGURE 3: EARLIEST START TIME FOR EACH TASK IN THE PIPELINE.....	22
FIGURE 4: PIPELINE COMPLETION TIME FOR DIFFERENT BITRATES	22
FIGURE 5: USE CASE APPLICATION PIPELINE	18



LIST OF TABLES

TABLE 1: INFRASTRUCTURE CONFIGURATION.....17



ABBREVIATIONS

ADS	ADA-PIPE scheduler
RPL	Resources preference list
TPL	Tasks preference list
IoT	Internet of things
CPU	Processor
MEM	Operating memory
STOR	Storage



1 INTRODUCTION

The *Internet of Things (IoT)* allows seamless cyber-physical integration of computing services in various application domains. The modern IoT data pipeline applications generate massive amounts of data and encompass complex processing tasks, including gathering, storing, and analysing raw-input data, which can overwhelm centralized Cloud-based computing infrastructures [1]. Without proper processing power for timely handling, the collected assets often remain unused, which in the absence of a meaningful exploitation expose more risks than value.

Recently, the so-called *computing continuum* that federates the Cloud services with emerging Fog and Edge resources, presented a relevant computing alternative for supporting the complex Big Data pipelines. However, eminent challenges in automating the processing of the IoT pipelines across the Computing Continuum still remain [2]. These include scheduling, deployment, and orchestration of the Big Data pipelines. Unfortunately, the heterogeneity of the computing continuum limits the possibility for utilizing resources provisioned by different providers, which further hinders the execution of the complex data pipelines with strict computing and network requirements.

Overall, the resource management across the continuum requires significant research effort, as the current data processing pipelines are dynamic, whereas traditional resource management strategies are static, which leads to inefficient pipelines scheduling and overly complex process deployment [3, 4]. Besides, with the advent of microservices architectures and containerization, the scheduling approaches must consider the proper data stream characteristics, facilitate networking connectivity, and provide optimized configuration of the complex pipeline with guaranteed scalability from the execution performance perspective.

Therefore, in this deliverable, we describe a novel matching-based data pipeline scheduling model, named *ADA-PIPE Scheduler (ADS)*, to address the problem of scheduling complex Big Data pipeline applications represented as directed acyclic graphs on heterogeneous computing continuum resources. ADS approaches this problem using matching theory principles involving two sets of players [5]:

- Pipeline steps (tasks) rank the continuum resources based on their pipeline processing and queuing time.
- Cloud, Fog, and Edge resources rank the tasks based on their pipeline transmission time.

The ADS two-sided matching-based scheduling model, integrated within the ADA-PIPE tool, assigns tasks to resources based on their mutual preferences, aiming to maximize the aggregated gains of the tasks and the resources in terms of pipeline processing, and data transfer and queuing times.

Hence, the main contributions of this deliverable are:

- A model for quantifying the data pipeline processing and queuing time tailored to asynchronous data exchange by utilizing the data queues.
- A ranking strategy including the transmission time in the resource-side ranking to localize the processing of users' data and meet the response time quality of service.



- A modified two-sided stable matching model based on the resource utilization for scheduling data pipeline on resources.
- An algorithm for scheduling of complex data pipeline on resources based on a two-sided stable matching model by considering the resource utilization.



2 STATE OF THE ART

This section reviews the state-of-the-art in task scheduling for Big Data pipelines in Cloud, Fog, and Edge computing environment. We therefore create taxonomy, based on the optimisation objectives utilized by the related approaches in the following categories:

- **Traffic-aware scheduling:** Arkian et al. [6] 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 workflow application. Tamiru et al. [7] designed an integrated scheduler in Kubernetes orchestration platform for multi-cluster computing environments. The authors modelled the resource requirement of the workload, the ingress traffic, and the resource capacity of each cluster to decide on an appropriate application deployment.
- **Cost-aware scheduling:** De Maio and Kimovski [8] investigated the potential of Fog computing for scheduling extreme data scientific workflows with the goal of optimizing processing time, reliability and user's monetary cost. Therefore, the authors modelled decomposition of the workflow with parallel tasks based on their dependencies, and proposed a multi-objective optimization-based scheduling method. Sharghivand et al. [4] proposed a two-sided matching solution to schedule IoT-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.
- **Latency-aware scheduling:** Menouer [9] designed a multi-criteria decision-analysis model to solve the scheduling problem. Therefore, the author presented the TOPSIS-based decision-making algorithm according to criteria such as the utilization of resources, alongside the processing time the workflow tasks and data transmission time. Fahs and Pierre [10] 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 workloads across replicas of an application (i.e., minimizing imbalance). Hoseiny et al. [11] designed a scheduling algorithm, based on an optimization model that is a weighted sum of overall pipeline processing time, energy consumption, and percentage of completed tasks with a given deadline. Lujic et al. [11] designed a model named SEA-LEAP as a data locality-aware scheduling method to minimize overall processing time for on-demand-analytics applications. The authors aimed to increase traffic safety by utilizing Edge computing infrastructure.
- **ADS contribution:** These works investigate the task scheduling as an optimization problem that minimizes the traffic, monetary cost or latency as main objectives, and neglect the processing, queuing and transmission times. We extend the related methods by researching a novel IoT data pipeline scheduling method based on a two-sided matching that considers the interests of the involved stakeholders to maximize the aggregated utility of tasks and resources to reach a stable equilibrium.



3 REQUIREMENTS ANALYSIS

The requirements for the big data pipelines scheduling of the DataCloud project are extracted from two main sources: analysis of the state-of-the-art literature (presented in Section 2) and the interviews with the business partners.

We therefore utilize the following requirements to guide the modelling process of the ADA-PIPE scheduler:

- **RQ-ADAPIPE-1:** Metric Specific Application Scaling.
- **RQ-ADAPIPE-4:** Limited Dynamic Scheduling.
- **RQ-ADAPIPE-5:** Requirement Definition per Task.
- **RQ-ADAPIPE-8:** QoS Guarantee for Tasks with Strict Deadlines.
- **RQ-ADAPIPE-10:** Avoidance of Highly Utilized Resources.
- **RQ-ADAPIPE-12:** Task Offloading on Edge and Mobile Devices.
- **RQ-ADAPIPE-14:** Data-Aware Pipeline Scheduling.
- **RQ-ADAPIPE-15:** Low Scheduling Overhead.

Besides, we also consider the following user stories, gathered with the AS-IS and TO-BE interviews described in Deliverable D1.1:

- **CER.ADA.US.03:** Schedule the tasks in transparent manner.
- **MOG.ADA.US.01:** Identify resources on the run that can support high stream data rates.
- **MOG.ADA.US.04:** Provide support for both online and offline scheduling.
- **JOT.ADA.US.04:** Move the data close to the resource.
- **JOT.ADA.US.08:** Maintain request processing latency below 200 ms.
- **BOSCH.ADA.US.02:** Define the resources where the tasks can be migrated.
- **BOSCH.ADA.US.04:** Define the architecture and computing requirements of each task.
- **TELLU.ADA.US.01:** Identify resources where side pipelines can be executed in parallel.

Therefore, from the list of the requirements, we can conclude that the ADS scheduler and the ADA-PIPE tool 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, the ADS scheduler 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 below 200ms.



4 SCHEDULING MODEL

4.1 PROBLEM DEFINITION

We represent a scheduling problem as a matching game using two disjoint sets of players:

- the tasks \mathbf{S} of the big data pipeline \mathbf{W} , and
- the resources registered in R-MARKET \mathbf{R} .

The game aims to match a task s_i in \mathbf{S} to a resource r_j in \mathbf{R} with sufficient capacity that optimizes the aggregated utility of application and resource provider. Therefore, our aim is to maximize the aggregated utility function by determining an equilibrium for stable matching of the tasks to appropriate resources.

We define the pipeline as directed acyclic graph (DAG) that comprises different paths from the producers to consumers, we inspect a valid matching sched for the tasks of every pipeline level $\mathbf{D}_{(d)}$ to maximize the aggregated utility function (see Section 3.2) of the tasks s_i in $\mathbf{D}_{(d)}$ and the resources $\mathit{sched} [d, s_i]$:

$$\max \left(\sum_{\substack{\forall s_i \in \mathcal{D}(d) \wedge \\ r_j \in \text{RPL}[s_i]}} U_s(s_i, r_j) + \sum_{\substack{\forall r_j \in \mathcal{R} \wedge \\ s_i \in \text{TPL}[r_j]}} U_r(s_i, r_j) \right)$$

Subject to:

A task s_i in in level $\mathbf{D}_{(d)}$ is matched to exactly one resource from its resources preference list $\mathbf{RPL}[s_i]$:

$$\mathit{sched}[d, s_i] \in \text{RPL}[s_i] \subseteq \mathcal{R} \wedge |\mathit{sched}[d, s_i]| = 1$$

A resource can execute multiple tasks that are members of its task preference list $\mathbf{TPL}[r_j]$ and within its capacity c_j considering the pipeline pip_{ui} :

$$\mathit{alloc}(r_j) \subseteq \text{TPL}[r_j] \subseteq \mathcal{S} \wedge \mathit{req}(s_i, \mathit{pip}_{ui}) \ll c_j$$

The matching does not contain blocking pairs of tasks and resources that prefer matching each other rather than their current assignments.

4.2 RANKING AND UTILITY MODEL

The model for matching tasks to resources uses a double ranking and utility strategy: task-sided and resource-sided.

4.2.1 Task-side ranking

The task side ranking orders the resources for a task s_i in a resource preference list $\mathbf{RPL}[s_i]$ based on the aggregated pipeline processing T_p and queuing times T_q :

$$T_{pq}(s_i, \mathit{pip}_{ui}, r_j) = T_p(s_i, \mathit{pip}_{ui}, r_j) + T_q(s_i, \mathit{pip}_{ui}, r_j)$$



4.2.2 Task-side utility

The task-side utility U_s represents the gain obtained by a task s_i scheduled on one of its preferred resources r_j in $RPL[s_i]$:

$$U_s(s_i, r_j) = \frac{|T_{pq}(s_i, \text{pip}_{ui}, r_j) - \text{LAST}_{T_{pq}}(RPL[s_i])|}{|\text{FIRST}_{T_{pq}}(RPL[s_i]) - \text{LAST}_{T_{pq}}(RPL[s_i])|}$$

where $\text{FIRST}_{T_{pq}}(RPL[s_i])$ and $\text{LAST}_{T_{pq}}(RPL[s_i])$ are lowest, respectively highest pipeline processing and queuing times of the resources in $RPL[s_i]$. The first resource in the preference list $RPL[s_i]$ provides the highest utility equals to one.

4.2.3 Resource-side ranking

The resource side ranking ranks the tasks for a resource r_j in a task preference list $TPL[r_j]$ based on the pipeline transmission time $T_c(s_u, \text{pip}_{ui}, s_i)$:

$$T_c(s_u, \text{pip}_{ui}, s_i) = \sum_{e \in \text{pip}_{ui}} t_c(s_u, e, s_i)$$

where $t_c(s_u, e, s_i)$ is the data transmission time for data e from task s_u to task s_i .

4.2.4 Resource-side utility

The resource-side utility presents the gain obtained by a resource r_j for executing one preferred task s_i in $TPL[r_j]$:

$$U_r(s_i, r_j) = \frac{|T_c(s_u, \text{pip}_{ui}, s_i) - \text{LAST}_{T_c}(TPL[r_j])|}{|\text{FIRST}_{T_c}(TPL[r_j]) - \text{LAST}_{T_c}(TPL[r_j])|}$$

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



5 SCHEDULING ALGORITHM

Algorithm 1 applies a matching-based heuristic method to schedule independent tasks per level of big data data pipeline on the available resources in R-MARKET.

After initializing the scheduling lists, (line 2), the algorithm loops over the dependency levels of the pipeline to rank the resources for independent tasks (line 5), and to rank the tasks for the resources (lines 6 - 12). After ranking, the algorithm attempts to find the appropriate matches based on the available resources (line 13). The matching function matches the task to one of its preferred resources, and then allocates the resource due to task's required capacity (despite of defining a fixed capacity for each resource in advance. 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 to an equilibrium while no task or resource have no other preferences than their current matched choices. Finally, the scheduling algorithm estimates the earliest start time and assigns the appropriate resource to the tasks (lines 14 - 16).

Algorithm 1 Scheduling algorithm.

```

Input:  $\mathcal{W} = (S, \mathcal{E}, \mathcal{Q}, S_{src}, S_{sank}, \mathcal{D})$   $\triangleright$  IoT data pipeline workflow
          $\mathcal{R} = \{r_j \mid 0 \leq j < \mathcal{N}_{\mathcal{R}}\}$   $\triangleright$  Cloud, Fog, and Edge resource set
          $\mathcal{L} = \{l_{kj} \mid 0 \leq k, j < \mathcal{N}_{\mathcal{R}}\}$   $\triangleright$  Channel set
Output:  $EST(s_i), sched[d, s_i]; \forall s_i \in \mathcal{W}$   $\triangleright$  Start time and scheduling of all tasks
1: function SCHEDULING( $\mathcal{W}, \mathcal{R}, \mathcal{L}$ )
2:   INITIALIZE( $sched$ )  $\triangleright$  Initialize scheduling list
3:    $d \leftarrow 1$ 
4:   for all  $d \leq \text{SIZE}(\mathcal{D})$  do
5:      $RPL \leftarrow \text{RANKRESOURCES}(\mathcal{W}, \mathcal{R}, \mathcal{L}, d)$   $\triangleright$  Rank resources for tasks of level  $d$  (invoke Alg. 2)
6:     if  $d = 1$  then  $\triangleright$  Rank tasks for resources
7:        $TPL \leftarrow \text{RANKTASKS}(\mathcal{W}, \mathcal{R}, \mathcal{L}, d, RPL, \emptyset)$   $\triangleright$  Invoke Alg. 3
8:     else
9:       if  $d \neq 1$  then
10:         $TPL \leftarrow \text{RANKTASKS}(\mathcal{W}, \mathcal{R}, \mathcal{L}, d, RPL, sched[d-1, *])$   $\triangleright$  Invoke Alg. 3
11:      end if
12:      end if
13:       $sched[d, *] \leftarrow \text{MATCH}(\mathcal{W}, \mathcal{R}, d, TPL, RPL)$   $\triangleright$  Match independent tasks to resources
14:      for all  $s_i \in \mathcal{D}(d) \wedge (s_u, s_i, pip_{ui}) \in \mathcal{E}$  do
15:         $EST(s_i) \leftarrow \max_{\forall (s_u, s_i, pip_{ui}) \in \mathcal{E}} (EST(s_u) + t_a)$ 
16:      end for
17:    end for
18:    return ( $EST, sched$ );
19: end function

```

The task-side ranking algorithm presented in Algorithm 2 receives as input the tasks of level, the data pipeline, and the set of resources and channels. The resource that guarantees a lower pipeline processing and queuing time receives a higher rank. The task-side ranking algorithm first initializes the resource preference lists for every task with the empty set in line 2. Thereafter, it filters the resources that do not satisfy the memory $MEM(s_i, pip_{ui})$ and storage $STOR(s_i, pip_{ui})$ requirements of a task (line 5). Afterward, it creates a list of tuples for each task that associates the pipeline processing and queuing time of a task on a resource (line 6). Finally, the algorithm sorts the resource preferences of each task based on its pipeline processing and queuing time in descending order in line 11.



Algorithm 2 Task-side ranking algorithm.

Input: $\mathcal{W} = (\mathcal{S}, \mathcal{E}, \mathcal{Q}, \mathcal{S}_{src}, \mathcal{S}_{snk}, \mathcal{D})$ \triangleright IoT data pipeline workflow
 $\mathcal{R} = \{r_j \mid 0 \leq j < \mathcal{N}_{\mathcal{R}}\}$ \triangleright Cloud, Fog, and Edge resource set
 $\mathcal{L} = \{l_{kj} \mid 0 \leq k, j < \mathcal{N}_{\mathcal{R}}\}$ \triangleright Channel set
 $d : 1 \leq d \leq \text{SIZE}(\mathcal{D})$ \triangleright Level d of workflow \mathcal{W}
Output: $\text{RPL}[s_i], \forall s_i \in \mathcal{W}$ \triangleright Resource preference lists of all tasks s_i

```

1: function RANKRESOURCES( $\mathcal{W}, \mathcal{R}, \mathcal{L}, d$ )
2:   INITIALIZE(RPL)  $\triangleright$  Initialize RPL
3:   for all  $s_i \in \mathcal{D}(d) \wedge (s_u, s_i, \text{pip}_{ui}) \in \mathcal{E} \wedge \text{qu}_{ui} \in \mathcal{Q}$ 
4:      $\hookrightarrow$  do  $\triangleright$  Rank resources for a task
5:     for all  $r_j \in \mathcal{R} \wedge l_{kj} \in \mathcal{L}$  do
6:       if  $(\text{MEM}(s_i, \text{pip}_{ui}) < \text{MEM}_j) \wedge (\text{STOR}(s_i, \text{pip}_{ui}) < \text{STOR}_j)$ 
7:          $\hookrightarrow$  then  $\triangleright$  Check constraints
8:          $\text{RPL}[s_i] \leftarrow \text{RPL}[s_i] \cup (r_j, T_{pq}(s_i, \text{pip}_{ui}, r_j))$ 
9:          $\hookrightarrow$  Add  $r_j$  and its  $T_{pq}$  to RPL
10:      end if
11:    end for
12:  end for
13:  return (RPL);
14: end function

```

Resource-side ranking algorithm presented in Algorithm 3, receives as input the resource preference lists computed in Algorithm 2, along with the big data pipeline, the resource set, the set of network channels \mathcal{L} , and subset of tasks in level $\mathcal{D}(d)$. Similarly, the algorithm initializes the task preference lists of a resource with the empty set in line 2. Afterward, each resource ranks the tasks in a preference list in line 8 based on its pipeline transmission time. Finally, the algorithm sorts the task preferences in descending order in line 12 based on the pipeline transmission time. Hence, the task with a lower pipeline transmission time receives a higher rank.

Algorithm 3 Resource-side ranking algorithm.

Input: $\mathcal{W} = (\mathcal{S}, \mathcal{E}, \mathcal{Q}, \mathcal{S}_{src}, \mathcal{S}_{snk}, \mathcal{D})$ \triangleright Data pipeline workflow
 $\mathcal{R} = \{r_j \mid 0 \leq j < \mathcal{N}_{\mathcal{R}}\}$ \triangleright Cloud, Fog, and Edge resource set
 $\mathcal{L} = \{l_{kj} \mid 0 \leq k, j < \mathcal{N}_{\mathcal{R}}\}$ \triangleright Channel set
 $d : 1 \leq d \leq \text{SIZE}(\mathcal{D})$ \triangleright Level d of workflow \mathcal{W}
 $\text{RPL}[s_i], \forall s_i \in \mathcal{D}(d)$ \triangleright Resource preference lists of tasks in level $\mathcal{D}(d)$
 $\text{sched}[d-1, *]$ \triangleright Matching/Scheduling list of level $d-1$
Output: $\text{TPL}[r_j], \forall r_j \in \mathcal{R}$ \triangleright Task preference lists of set of resources \mathcal{R}

```

1: function RANKTASKS( $\mathcal{W}, \mathcal{R}, \mathcal{L}, d, \text{RPL}, \text{sched}$ )
2:   INITIALIZE(TPL)  $\triangleright$  Initialize TPL
3:   for all  $s_i \in \mathcal{D}(d) \wedge (s_u, s_i, \text{pip}_{ui}) \in \mathcal{E}$  do  $\triangleright$  Rank tasks for a resource
4:     if  $d \neq 1$  then
5:        $r_k \leftarrow \text{sched}[d-1, s_u]$   $\triangleright$  Get upstage matched resource
6:     end if
7:     for all  $r_j \in \text{RPL}[s_i] \wedge l_{kj} \in \mathcal{L}$  do
8:        $\text{TPL}[r_j] \leftarrow \text{TPL}[r_j] \cup (s_i, T_c(s_u, \text{pip}_{ui}, s_i))$ 
9:        $\hookrightarrow$  Add  $s_i$  and its  $T_c$  to TPL
10:    end for
11:  end for
12:  for all  $(r_j \in \mathcal{R}) \wedge (\text{TPL}[r_j] \neq \emptyset)$  do
13:     $\text{TPL}[r_j] \leftarrow \text{Sort}_{T_c}(\text{TPL}[r_j])$   $\triangleright$  Sort based on  $T_c$ 
14:  end for
15:  return (TPL);
16: end function

```



6 ADA-PIPE SCHEDULER TRACE EXAMPLE

Figure 1 illustrates an example of using Algorithm 1 in the following steps:

1. Figure 1(a) exemplifies the input for scheduling four inter-dependent tasks on a set of resources r_0, r_1, r_2 .
2. Figure 1(b) denotes three independent levels of big data pipeline. The levels are computed by considering the downward levels starting from the producers of data and ending with consumers of data. In this manner, the task s_0 is in the first level since it consumes data received from producer(s). The second level consists of all the tasks dependent on the first-level tasks, which are tasks s_1 and s_2 in our example. Since s_3 is dependent on tasks s_1 and s_2 , then third level consists of just one task.
3. Figure 1c continues with the ranking of the tasks **RPL** and resources **TPL** for each level. In resource preference list of task s_0 , r_2 has the highest rank because it provides the lowest pipeline processing and queuing time. From the other side, there is just one task in task preference list of r_2 , therefore, r_2 matches to s_0 .
4. Figure 1d shows that r_2 and r_1 matches to tasks s_1 and s_2 , respectively, because they rank these tasks as the highest ones with the lowest pipeline transmission times.
5. Figure 1e depicts the last level's task s_3 that matches to its first preference r_0 , which provides lowest pipeline processing and queuing time.
6. Figure 1f shows the scheduled resources and the earliest estimated start time **est** for the tasks of all levels.

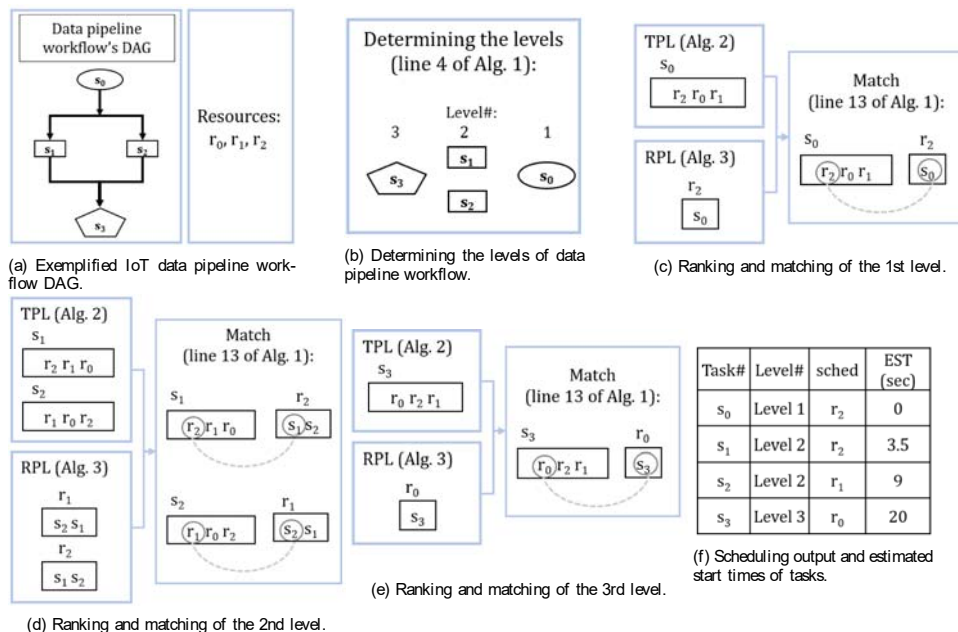


Figure 1: ADA-PIPE Scheduler trace example



7 PRELIMINARY EVALUATION

We implemented the ADS matching-based big data pipeline scheduling in Python v.3.9 using the matching library [13]. Then, we integrated our customized scheduler in Kubernetes orchestrator tool v1.21 by utilizing the Python client library for Kubernetes v17.17 [14]. The script required to run the ADS model is available in the DataCloud code repository¹.

7.1 INFRASTRUCTURE SETUP

We deploy ADA-PIPE as a RESTful service in the Carinthian Computing Continuum C³ infrastructure testbed, providing a rich set of resources across the Cloud, Fog, and Edge layers, illustrated in Table 1.

Table 1: Infrastructure configuration

	Cloud		Fog		Edge				
	AWS -Virginia t2.xlarge	Google n2 -standard	A1/Exoscale large instance	A1/Exoscale medium instance	Edge large instance	Edge medium instance	NvJ	RP14	RP13
CPU (#cores)	4	4	4	2	24	16	4	4	4
MEM (GB)	16	16	8	4	32	16	4	4	1
STOR (GB)	8	8	10	10	32	32	16	16	16
BW (Mbit s ⁻¹)	100	870	840	839	920	900	450	800	328
LAT (ms)	101	23	11.5	11.9	0.3	0.3	1	0.4	1

We utilized the Prometheus operator monitoring system v0.45.0² for monitoring the Kubernetes services and deployment alongside the network traffic, bandwidth, latency and resources utilization rates.

We utilized the asynchronous message queue architecture KubeMQ v2.2.1³ to implement data exchange between tasks. KubeMQ provides a message queue for every pipeline task to store messages from its upstage tasks.

7.2 RELATED WORK COMPARISON

We conduct the performance comparisons against three state-of-the-art approaches:

- Heterogeneous Earliest Finish Time -- only Cloud (HEFT-oC) schedules all tasks on the Cloud and selects the proper Cloud instances using a bottom ranking approach to optimize the pipeline completion time [15].
- Kubernetes container scheduling strategy (KCSS) schedules all tasks on Edge and Fog resources using the Topsis algorithm based on multiple criteria such as pipeline

¹ <https://github.com/SiNa88/example-kubernetes-scheduler>

² <https://github.com/prometheus-operator/prometheus-operator>

³ <https://github.com/kubemq-io/kubemq-community/releases/tag/v2.2.10>



completion time, the number of Docker images and the number of running containers on each resource [9].

- SEA-LEAP minimizes data pipeline completion time, and identifies an Edge resource closer to the dataset distributed on multiple Fog and Edge resources [12].

7.3 USE CASE APPLICATION

We selected a representative traffic management system case study following road safety inspection concerns. We represent this application as a DAG of six tasks depicted in Figure 2. Every task communicates with its upstage tasks through a message queuing system:

1. Encoding encodes the raw video in various bitrates and resolutions.
2. Framing utilizes OpenCV to produce still frames.
3. Inference creates an inference model with high accuracy.
4. Dataset storage provides the stored records for other stages.
5. Training updates and retrains the multi-class classification model with a high accuracy.
6. Packaging and delivery provide the detected signs in the format required by the drivers.

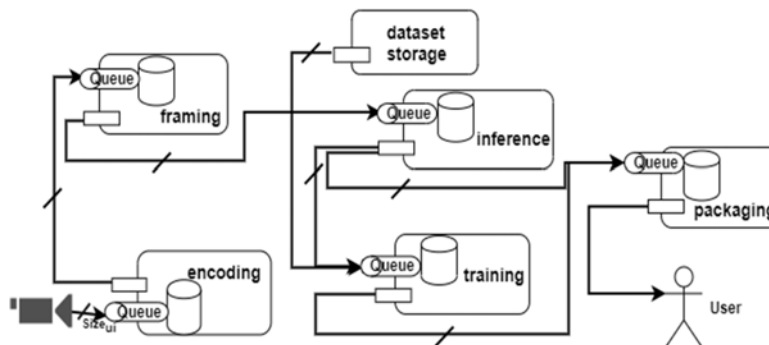


Figure 2: Use case application pipeline

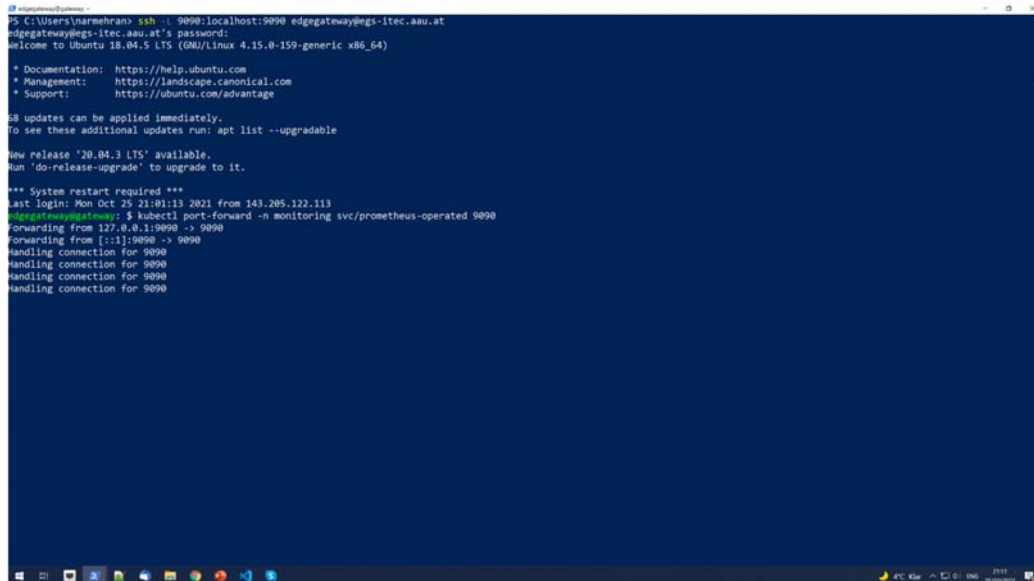
7.4 MONITORING

We collect the computing and networking metrics with the Prometheus monitoring tool⁴ in the following manner:

1. Get remotely connected to the utilized devices.
2. Redirect the traffic received on port number 9090 to the local machine where the data can be analysed. The following command redirects the traffic from a specific port of the Kubernetes master node to your local machine:

⁴ <https://prometheus.io/docs/introduction/overview/>

```
ssh -L 9090:localhost:9090 edgegateway@egs-itec.aau.at
```



```

PS C:\Users\Narmehran> ssh -L 9090:localhost:9090 edgegateway@egs-itec.aau.at
edgegateway@egs-itec.aau.at's password:
Welcome to Ubuntu 18.04.5 LTS (GNU/Linux 4.15.0-159-generic x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:    https://landscape.canonical.com
 * Support:       https://ubuntu.com/advantage

8 updates can be applied immediately.
To see these additional updates run: apt list --upgradable

New release '20.04.3 LTS' available.
Run 'do-release-upgrade' to upgrade to it.

*** System restart required ***
last login: Mon Oct 25 21:01:13 2021 from 143.205.122.113
edgegateway@egs-itec.aau.at:~$ kubectl port-forward -n monitoring svc/prometheus-operated 9090
Forwarding from 127.0.0.1:9090 -> 9090
Forwarding from [::]:9090 -> 9090
Handling connection for 9090
Handling connection for 9090
Handling connection for 9090
Handling connection for 9090

```

3. We Set up a kube-latency⁵ container on a worker node to measure bandwidth and latency of the Kubernetes network.
4. We use the following commands to gather various monitoring data:
 - physical link latency between every two nodes will be calculated:
`sort_desc(avg(ping_durations_s{quantile='0.99'})by(source_node_name,dest_node_name))`
 - the amount of data (in terms of bytes) received by any node during the last hour:
`sort_desc(rate(node_network_receive_bytes_total{device="eth0"}[1h])) * 8 / 1024 / 1024`
 - the amount of data received by a special container:
`sort_desc(container_network_receive_bytes_total{pod=~"name.*"})`

7.5 SCHEDULER EXECUTION

The scheduler executes in a main loop. The main loop of the scheduler waits for a new pipeline task comprising a container of the use case application in the *pending* state alongside a specification assigned by the ADS.

Then, it sets the pending pod to the node selected based on the ADS algorithm. Afterward, one by one the *pending* pods will be deployed and executed on the selected devices, as shown in the figure below:

⁵ <https://github.com/simonswine/kube-latency>



```
edgegateway@gateway: ~/Documents/Name/project$ kubectl get pods
NAME                                READY   STATUS    RESTARTS   AGE
pingtest-64f9cb6b84-2vxs4          1/1     Running   4           158d
pingtest-64f9cb6b84-cxg4h          1/1     Running   1           129d
pingtest-64f9cb6b84-dzt7n          1/1     Running   5           158d
pingtest-64f9cb6b84-tb7ms          1/1     Running   0           19d
pingtest-64f9cb6b84-wqt1q          1/1     Running   0           111d
test-kl-kube-latency-t95xp          1/1     Running   10          158d
test-kl-kube-latency-t98gd          1/1     Running   1535        160d
test-kl-kube-latency-wwk72          1/1     Running   12          160d
test-kl-kube-latency-xmwfr          1/1     Running   11          158d
test-kl-kube-latency-zssnf          1/1     Running   12          158d
edgegateway@gateway: ~/Documents/Name/project$
```

The output of the scheduling plan shows the nodes on which the nodes will be scheduled such as gateway device in C³ testbed:

```
edgegateway@gateway: ~/Documents/Name/project$ python3.9 scheduler.py
-----
scheduling pod 33training-56f68c896c-76j6s
33training-56f68c896c-76j6s scheduled on gateway
-----
scheduling pod 11framing-f6c9884f4-k6fhv
11framing-f6c9884f4-k6fhv scheduled on gateway
-----
scheduling pod 00encoding-6445546546-t6kpm
00encoding-6445546546-t6kpm scheduled on gateway
```

After the execution of scheduler, the pods status changes to *Running* state:

```
edgegateway@gateway: ~/Documents/Name/project$ kubectl get pods
NAME                                READY   STATUS    RESTARTS   AGE
00encoding-6445546546-t6kpm         1/1     Running   210        17h
11framing-f6c9884f4-k6fhv           1/1     Running   210        17h
2inferencehigh-568d84d96-v948j      1/1     Running   25417      105d
33training-56f68c896c-76j6s         1/1     Running   210        17h
pingtest-64f9cb6b84-2vxs4           1/1     Running   4           173d
pingtest-64f9cb6b84-4fcng           1/1     Running   1           173d
pingtest-64f9cb6b84-cxg4h           1/1     Running   1           144d
pingtest-64f9cb6b84-dzt7n          1/1     Running   5           173d
pingtest-64f9cb6b84-tb7ms           1/1     Running   0           34d
pingtest-64f9cb6b84-wqt1q           1/1     Running   0           126d
test-kl-kube-latency-t95xp           1/1     Running   10          172d
test-kl-kube-latency-t98gd           1/1     Running   1535       174d
test-kl-kube-latency-wwk72           1/1     Running   12          174d
test-kl-kube-latency-xmwfr           1/1     Running   11          172d
test-kl-kube-latency-zssnf           1/1     Running   12          172d
```

7.6 EVALUATION SCENARIO

We evaluated the ADA-PIPE scheduler compared to the related SEA-LEAP, KCSS, and HEFT-oC methods using the video pipeline for traffic sign classification. We selected a raw 9 seconds long video with size of 45 MB.

We designed a set of experiments according to the pipeline characteristics related to the encoding bitrate. The encoding bitrate experiments investigate the impact of CPU requirements for encoding the raw video segment. We considered five encoding bit rates of 200, 1500, 3000, 6500, 20000 kbps. We further considered two machine learning models with



70% and 90% accuracy for the inference and training tasks with different CPU requirements. We fixed the size of the data element to 2560 kbytes.

7.7 RESULTS

Figure 3 shows that by increasing the number of tasks while providing the same amount of computing and storage resources, the task utility reduces. The reason is that ADS matches the higher-ranked tasks to highly provided bandwidth resources such as the Edge instances. Therefore, there are just a few tasks matched to their lower-ranked resources that do not impact on the whole utility. From the resource-side perspective, the resource utility increases and ADS matches more tasks to resources because of the larger available resource capacities. However, when the number of tasks reaches to thirty because of the limited resource capacities, the resource-side utility reduces.

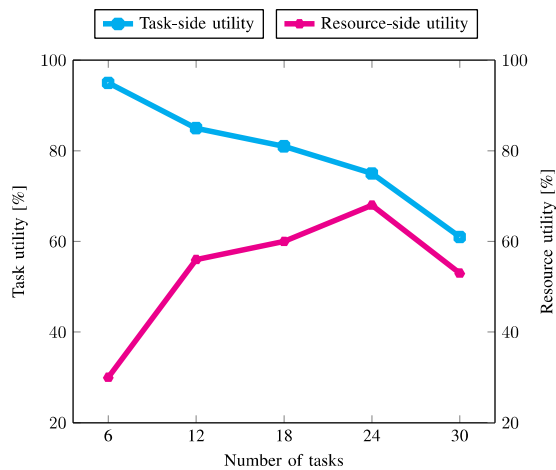


Figure 3: Task and resources utility for different number of tasks

Figure 4 depicts the earliest start time of pipeline tasks. As observed, by utilizing the queue as a temporary storage between tasks, especially between inference and training tasks, alongside training and packaging tasks, we reach improvement in terms of the earliest start time of the tasks.

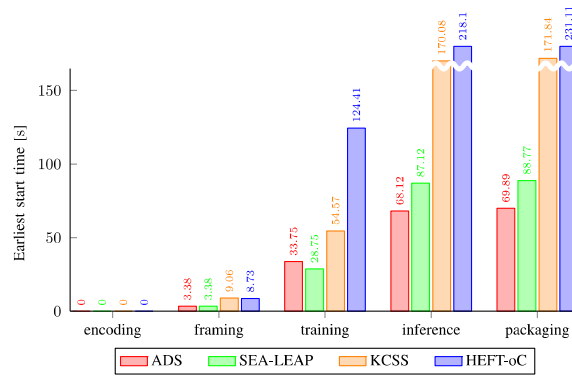


Figure 4: Earliest start time for each task in the pipeline

Figure 5 denotes that ADS reduces the completion time by 14 compared to SEA-LEAP because ADS selects the Edge instance that is closer to the sources and the dataset that is located on the Fog resources.

In addition, ADS reduces the completion time, on average, 56% compared to KCSS, which tends to schedule the tasks on the Cloud instances without considering the transmission time to reach the Cloud; however, this strategy is not proper for deadline-constrained applications.

In other words, unlike the related methods, ADS searches for tradeoff by scheduling on resources with lower pipeline transmission time to the data sources and higher computational speed; therefore, it improves the completion time by reducing the pipeline transmission time and the pipeline processing and queuing time that leads to reduced pipelines's completion time.

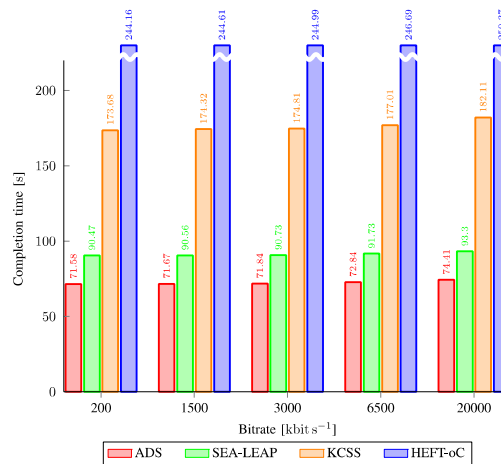


Figure 5: Pipeline completion time for different bitrates



8 CONCLUSIONS

In this deliverable, we introduced ADS, a matching-based scheduling method integrated within the DataCloud infrastructure that considers the pipeline processing and transmission times for scheduling the Big Data data pipeline on the computing continuum. ADS uses a two-sided matching algorithm to schedule the tasks based on their pipeline processing requirements, and the available computation and communication capabilities from the resource provider perspective.

To solve the scheduling and provisioning problem, ADS determines the dependency levels of pipeline's DAG. Afterward, ADS approaches this problem using matching theory principles involving two sets of players. First, every level of pipeline ranks the continuum resources based on their pipeline processing and queuing times. Second, Cloud, Fog, and Edge resources rank the tasks of the level based on their pipeline transmission time. After ranking, ADS matches the tasks to resources in the case they have enough capacity until all the data pipeline tasks are matched to the resources.

We have conducted preliminary evaluation of ADS and integrated it with Kubernetes and Prometheus. The results also show that ADS is more energy-efficient than other related work approaches. In other words, since ADS inspects the Cloud, Fog, and Edge resources that have enough computational capabilities and can lower pipeline transmission time, we achieved 14 - 68% lower completion time.

In addition, we also provide in this deliverable a full user manual, case study application and source code of the scheduler.

In the future, we plan to further explore a heuristic-, matching-based pipeline scheduling in the computing continuum, and integrate the scheduler with SIM-PIPE, R-MARKET and DEP-PIPE.



9 REFERENCES

- [1] Sunil Singh Samant, Mohan Baruwal Chhetri, Quoc Bao Vo, Ryszard Kowalczyk, and Surya Nepal. Towards end-to-end qos and cost-aware resource scaling in cloud-based iot data processing pipelines. In 2018 IEEE International Conference on Services Computing (SCC), pages 287–290. IEEE, 2018.
- [2] Mutaz Barika, Saurabh Garg, Albert Y Zomaya, Lizhe Wang, Aad Van Moorsel, and Rajiv Ranjan. Orchestrating big data analysis workflows in the cloud: research challenges, survey, and future directions. *ACM Computing Surveys (CSUR)*, 52(5):1–41, 2019.
- [3] Sarder Fakhurul Abedin, Md Golam Rabiul Alam, SM Ahsan Kazmi, Nguyen H Tran, Dusit Niyato, and Choong Seon Hong. Resource allocation for ultra-reliable and enhanced mobile broadband iot applications in fog network. *IEEE Transactions on Communications*, 67(1):489–502, 2018.
- [4] Nafiseh Sharghivand, Farnaz Derakhshan, Lena Mashayekhy, and Leyli Mohammad Khanli. An edge computing matching framework with guaranteed quality of service. *IEEE Transactions on Cloud Computing*, pages 1–1, 2020.
- [5] Narges Mehran, Dragi Kimovski, and Radu Prodan. A two-sided matching model for data stream processing in the cloud – fog continuum. In 2021 IEEE/ACM 21st International Symposium on Cluster, Cloud and Internet Computing (CCGrid), pages 514–524. IEEE, 2021.
- [6] Hamidreza Arkian, Guillaume Pierre, Johan Tordsson, and Erik Elmroth. Model-based stream processing auto-scaling in geo-distributed environments. In ICCCN 2021-30th International Conference on Computer Communications and Networks, 2021.
- [7] Mulugeta Tamiru, Guillaume Pierre, Johan Tordsson, and Erik Elmroth. mck8s: An orchestration platform for geo-distributed multi-cluster environments. In ICCCN 2021-30th International Conference on Computer Communications and Networks, 2021.
- [8] Vincenzo De Maio and Dragi Kimovski. Multi-objective scheduling of extreme data scientific workflows in fog. *Future Generation Computer Systems*, 106:171 – 184, 2020.
- [9] Tarek Menouer. Kcss: Kubernetes container scheduling strategy. *The Journal of Supercomputing*, 77(5):4267–4293, 2021.
- [10] Ali J Fahs and Guillaume Pierre. Tail-latency-aware fog application replica placement. In International Conference on Service-Oriented Computing, pages 508–524. Springer, 2020.
- [11] Farooq Hoseiny, Sadoon Azizi, Mohammad Shojafar, Fardin Ahmadiazar, and Rahim 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), pages 1–6, 2021.
- [12] Ivan Lujic, Vincenzo De Maio, Srikumar Venugopal, and Ivona Brandic. Sea-leap: Self-adaptive and locality-aware edge.
- [13] Henry Wilde, Vincent Knight, and Jonathan Gillard. Matching: A python library for solving matching games. *Journal of Open Source Software*, 5(48):2169, 2020.



[14] Michael Chima Ogbuachi, Anna Reale, Peter Suskovics, and Benedek ´ Kovacs. Context-aware kubernetes scheduler for edge-native applications on 5g. *Journal of Communications Software and Systems*, 16(1):85–94, 2020.

[15] Haluk Topcuoglu, Salim Hariri, and Min-you Wu. Performance-effective and low-complexity task scheduling for heterogeneous computing. *IEEE transactions on parallel and distributed systems*, 13(3):260–274, 2002.

