Grant Agreement No.: 101070473

Type of action: HORIZON-RIA

Topic: HORIZON-CL4-2021-DATA-01-05

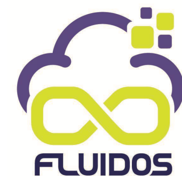
Call: HORIZON-CL4-2021-DATA-01

D4.1 INTENT-BASED DECENTRALISED FLUIDOS CONTINUUM

V1

Revision: v.1.0

Work package	WP 4
Task	Task 4.1, 4.2 and 4.3
Due date	30/11/2023
Submission date	30/11/2023
Deliverable lead	IBM
Version	0.17
Authors	Liubov Nedoshivina, Stefano Braghin, Ambrish Rawat (IBM) Alberto Robles Enciso, Alejandro Molina Zarca, Jose Manuel Bernabe (UMU) Marco Zambianco (FBK) Alessio Sacco (POLITO) Nicolas Kourtellis (TID) George Kornaros (HMU)
Reviewers	Silvio Cretti, Marco Zambianco (FBK), Marcello Coppola (STM), Amjad Majid (MAR)

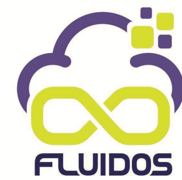


Abstract	The deployment of workloads in a distributed environment is known to be complex tasks, and it is even more complex when the system is expected to satisfy requirements in a dynamic context balancing factors like performance, resource consumption, and infrastructure and service costs. Thus, in a scenario like the one targeted by FLUIDOS. This deliverable presents the results of the main activities of WP4 for establishing when an application should leverage the continuum. These activities are grounded on formal definitions of user requirements in the form of intents. We further illustrate various research efforts leading to use-case driven algorithm for optimal workload placement using various techniques, including linear optimization, traditional machine learning, and deep neural networks. Finally, we present the initial work on how to foster collaboration between participants of the continuum by improving security and privacy formal guarantees with respect to distributed model training.
Keywords	FLUIDOS, Meta Orchestration, Kubernetes scheduling

Document Revision History

Version	Date	Description of change	List of contributor(s)
0.1	15/01/2018	1st version of the template for comments	Margherita Facca (Martel)
0.2	01/11/2023	1st draft of D4.1 document	Liubov Nedoshivina, Ambrish Rawat (IBM)
0.3	05/11/2023	Incorporated UMU's contribution	Liubov Nedoshivina (IBM), Alberto Robles Enciso (UMU),
0.4	09/11/2023	Incorporated HMU's and POLITO's contributions	Stefano Braghin, Liubov Nedoshivina (IBM), George Kornaros (HMU), Alessio Sacco (POLITO)
0.5	10/11/2023	Incorporated FBK's contribution	Liubov Nedoshivina, Stefano Braghin (IBM), Marco Zambianco (FBK)
0.6	12/11/2023	Incorporated TID's contribution	Liubov Nedoshivina, Stefano Braghin (IBM), Nicolas Kourtellis (TID)
0.7	13/11/2023	Expanded model-based orchestration	Liubov Nedoshivina, Stefano Braghin (IBM)
0.8	14/11/2023	Expansion of Intent definitions	Liubov Nedoshivina, Stefano Braghin (IBM)
0.9	15/11/2023	Update of 2.6.1	Alejandro Molina Zarca, Jose Manuel Bernabe (UMU)
0.10	15/11/2023	Improved introduction	Liubov Nedoshivina, Stefano Braghin (IBM)
0.11	15/11/2023	Expanded MSPL definition	Stefano Braghin (IBM)
0.12	16/11/2023	Expanded model-based architecture	Liubov Nedoshivina, Stefano Braghin (IBM)
0.13	17/11/2023	Update of Policy-based meta-orch.	Jose Manuel Barnabe (UMU)
0.14	20/11/2023	Update based on FBK review	Liubov Nedoshivina, Stefano Braghin (IBM), Marco Zambianco, Silvio Cretti (FBK)

Funded by Horizon Europe Framework Programme of the European Union



0.15	21/11/2023	Update based on STM review	Liubov Nedoshivina, Stefano Braghin (IBM), Marcello Coppola (STM)
0.16	22/11/2023	Improved abstract and executive summary	Liubov Nedoshivina, Stefano Braghin (IBM)
0.17	27/11/2023	Integrated MRT comments from MAR	Amjad Majid (MAR), Stefano Braghin (IBM)

Funded by Horizon Europe
Framework Programme of the European Union

DISCLAIMER

The information, documentation, and figures available in this deliverable are written by the "Flexible, scaLable and secUre decentralIzeD Operationg" (FLUIDOS) project's consortium under EC grant agreement 101070473 and do not necessarily reflect the views of the European Commission.

The European Commission is not liable for any use that may be made of the information contained herein.

COPYRIGHT NOTICE

© 2022 - 2025 FLUIDOS Consortium

Project co-funded by the European Commission in the Horizon Europe Programme		
Nature of the deliverable:	**R***	
Dissemination Level		
PU	Public, fully open, e.g., web	✓
SEN	Sensitive, limited under the conditions of the Grant Agreement	
Classified R-UE/ EU-R	EU RESTRICTED under the Commission Decision No2015/ 444	
Classified C-UE/ EU-C	EU CONFIDENTIAL under the Commission Decision No2015/ 444	
Classified S-UE/ EU-S	EU SECRET under the Commission Decision No2015/ 444	

** R: Document, report (excluding the periodic and final reports)*

DEM: Demonstrator, pilot, prototype, plan designs

DEC: Websites, patents filing, press & media actions, videos, etc.

DATA: Data sets, microdata, etc

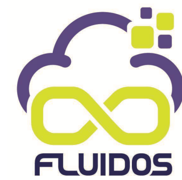
DMP: Data management plan

ETHICS: Deliverables related to ethics issues.

SECURITY: Deliverables related to security issues

OTHER: Software, technical diagram, algorithms, models, etc.

Funded by Horizon Europe Framework Programme of the European Union



EXECUTIVE SUMMARY

This deliverable summarizes the activities of Work Package 4, specifically Task 4.1 Task 4.2, and Task 4.3. The deliverable opens, in Section 1, contextualizing the activities of the tasks within both the overall project landscape, and with respect to the research questions that are driving innovation within the work package.

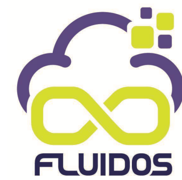
After that, the deliverable focuses in Section 2, on the formal and informal definition of intent, i.e. the conceptual mechanism that allows users to characterize formal and informal constraints for workload execution. These constraints encompass both traditional resource requirements, including geographical location, and more advanced, high level, concepts such as performance requirements, compliance specifications, cost constraints, and more. We present a series of languages, both from the state of the art and proposals from FLUIDOS, for the definition of such constraints. These languages assume the existence of a conceptual mapping between them. A first version of this mapping exists. It has been created as part of FLUIDOS activities, specifically the ones described in D3.1.

The deliverable illustrates concrete research outcomes, consisting of new algorithms, methodologies – both presented in Section 3.1 -, and concrete proof of concept implementations – illustrated in Section 3.2 - to satisfy the workload requirements a user can express through the previously introduced intent languages.

The algorithms and methodologies presented focus on cost-oriented workload orchestration, refer to Section 3.1.1, latency-aware orchestration, refer to Section 3.1.2, high-level policy meta-orchestration, refer to Section 3.1.3, and finally a purely ML-based meta orchestration, refer to Section 3.1.4. These different approaches are presented with various levels of detail and experimental evaluation depending on the maturity of the research. Nonetheless, they all demonstrate the feasibility of workload orchestration, or meta-orchestration, within the FLUIDOS continuum.

Following the presentation of the more theoretical components of the work conducted within WP4, the deliverable focuses on the artifacts that are being provided to the community and to the users of the open calls. These artifacts are described in Section 3.2 where the two main approaches, policy-based and ML-based, are also contextualized within the FLUIDOS

Funded by Horizon Europe
Framework Programme of the European Union



architecture referred by the publicly available REAR protocol architecture[1], and in deliverable D2.1.

Finally, Section 4 presents a series of theoretical work aimed as foundational for the subsequent activities of Task 4.3. Namely, Section 4.2 and Section 4.3 present novel approaches to the application of privacy-enhancing technologies (PETs) to well-known techniques to train ML models in distributed settings, thus Federated Learning (FL).

[1] https://github.com/fluidos-project/REAR

Funded by Horizon Europe Framework Programme of the European Union

TABLE OF CONTENTS

1 Introduction *13*

2 Intent *15*

2.1 Medium-level Security Policy Language **15**

2.2 INTEL Intent policy language **16**

2.3 Topology and Orchestration Specification for Cloud Applications **17**

2.4 Network Intent Languages **18**

3 FLUIDOS Node Meta-Orchestrator *19*

3.1 Algorithms for Orchestration in Cloud to Edge Continuum **19**

3.1.1 Cost-based orchestration 19

3.1.2 Latency-aware orchestration 27

3.1.3 Policy-based Meta-orchestration 32

3.1.4 Model based meta-orchestration 35

3.1.5 Orchestration on the Edge 38

3.2 Architecture **41**

3.2.1 MSPL-based Meta-Orchestration 42

3.2.2 Model-based Meta-Orchestration 45

4 Privacy & Security of Model-based Intent-driven Meta-Orchestration *48*

4.1 Background **49**

4.1.1 Differential Privacy 49

4.1.2 Federated Learning 50

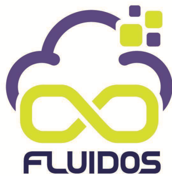
4.1.3 Federated learning with Differential Privacy 51

4.1.4 Hierarchical Federated Learning 51

4.1.5 Asynchronous Federated Learning 52

© 2022-2025 FLUIDOS

Funded by Horizon Europe Framework Programme of the European Union



4.1.6 Buffered Asynchronous Federated Learning 53

4.2 Hierarchical Federated Learning meets Differential Privacy 53

4.2.1 Impact on Scalability 54

4.2.2 Impact on Privacy 54

4.2.3 Proposed Algorithm 55

4.2.4 Evaluation 56

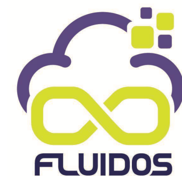
4.3 Asynchronous Federated Learning with Local Differential Privacy 59

4.3.1 Proposed Algorithm 60

4.3.2 Evaluation 61

References 65

Funded by Horizon Europe
Framework Programme of the European Union



LIST OF FIGURES

Figure 1 Relationship between technical WPs 14

Figure 2 Relationship between FLUIDOS MSPL and prior work. 16

Figure 3 An example of intent manifest according to Intel's definition. 17

Figure 4 A simple "Hello World" for TOSCA. 18

Figure 5 Feasibility-optimality trade-off. a) Feasibility percentage; b) Total deployment cost for each application. 25

Figure 6 Realistic case with increasing number of domains. a) Feasibility percentage; b) Average deployment cost for each application. 25

Figure 7 Latency (box plot) and success rate (blue curve) for all testing scenarios. 31

Figure 8 CDF of converge time for the three scenarios and comparison with alternative Latency-Aware Schedulers (LAK). 32

Figure 9 Meta-Orchestration Process 33

Figure 10 Allocation Process 34

Figure 11 Orchestrating Edge Applications 39

Figure 12 Meta-orchestrator conceptual architecture 42

Figure 13 Example of MSPL-based orchestration algorithm. 43

Figure 14 Example of an MSPL policy. 44

Figure 15 High level K8S Operator Paradigm 45

Figure 16 High Level FLUIDOS Model-based Meta-Orchestrator 46

Figure 17 Example of pod manifest with explicit intent 46

Figure 18 Architectural overview of our attack setup. Locally learnt gradients are perturbed with noise to provide DP guarantees. This ensures the gradients viewed by the aggregator are less noise. Adversarial nodes can be present at level 0 or 1 54

Figure 19 Modified FL mechanism with zonal privacy. 55

Figure 20 We plot the validation accuracy of training with DP for 3 scenarios: (i) central DP, (ii) local DP, and (iii) the proposed hierarchical DP. 58

© 2022-2025 FLUIDOS

Funded by Horizon Europe
Framework Programme of the European Union

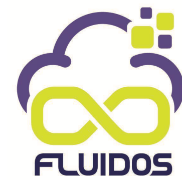


Figure 21 Privacy expenditure across datasets and DP methods. *59*

Figure 22 We plot the validation accuracy as a function of privacy for 3 scenarios using HDP: (i) K = 100, (ii) K=200, and (iii) K = 300. .... *59*

Figure 23 Local Differential Privacy in Buffered AFL (FedBuff+LDP) *61*

Figure 24 Main task accuracy in buffered async FL with LDP in different communication trips. *63*

Figure 25 Main task accuracy in buffered async FL with LDP for three datasets with varying privacy budgets (lower provides better privacy). *64*

Figure 26 Average ± standard deviation number of communication trips (in thousands) to reach a target accuracy in the three datasets with = 8 and K = 10. Standard deviation is computed over 5 random runs of each setting with different seeds. *64*

Funded by Horizon Europe Framework Programme of the European Union

LIST OF TABLES

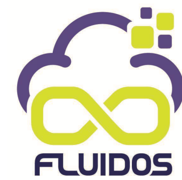
Table 1 Applications' microservices requirements. 24

Table 2 Execution time (sec) with increasing number of applications to deploy. 26

Table 3 Execution time (sec) with increasing number of domains. 27

© 2022-2025 FLUIDOS

Funded by Horizon Europe
Framework Programme of the European Union

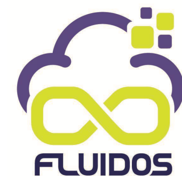


ABBREVIATIONS

IP	Internet Protocol
TCP	Transmission Control Protocol
AI	Artificial Intelligence
ML	Machine Learning
DaaS	Desktop as a Service
QoE	Quality of Experience
SLO	Service Level Objectives
ILP	Integer Linear Programming
MIPS	Million Instructions Per Second
MSPL	Medium-level Security Policy Language
GA	Genetic Algorithm
RL	Reinforcement Learning
K8S	Kubernetes
MF	Matrix Factorization
DP	Differential Privacy
LDP	Local Differential Privacy
LAK	Latency-Aware Kubernetes
PET	Privacy Enhancing Technology

© 2022-2025 FLUIDOS

Funded by Horizon Europe
Framework Programme of the European Union



1 INTRODUCTION

The creation of a cloud-to-edge continuum, as the one envisioned by FLUIDOS requires several complex techniques to be coordinated at several levels. Some of the activities to achieve this ambitious goal, specifically the mechanics, or the "how" to establish a continuum, are addressed in other work packages. On the other hand, the activities of Work Package 4 focus on identifying what a user would require leveraging resources within a cloud-to-edge continuum. This is a far from trivial task because it generally assumes a complete knowledge and understanding of users' requirements. This information is generally not available, or there could be some potential issues in what a user specifies and what the system is able to provide, also considering potential misunderstanding of system capabilities and actual workload requirements from the user point of view. Therefore, WP4 focuses on exploring and evolving methodologies to allow users to specify workload requirements at various levels of abstraction. Traditionally, orchestration engines request a specific quantity of resources (e.g., number of CPUs or latency), a quantity range (e.g., min/max number of CPUs) or not specifying them at all, to support the appropriate placement of workloads. The existing variety of implementations aims to solve the issue of optimal workload deployment by means of rule-based approaches, linear programming methods, and more. Machine Learning (ML) algorithms are accordingly employed by container orchestration systems for behaviour modelling and prediction of multi-dimensional performance metrics. Note how these tasks complement the work performed in WP6, where more attention is given to specific requirements and metrics, namely cost and energy.

Once a user can specify workload and requirements there are more challenges to address. That is, the identification of the resources required for the system to satisfy the requirements for a specific workload, considering both locally available resources and those provided by other participants to the continuum. Again, this is a challenge as the optimal placement of a workload within a continuum might change as the continuum evolves. Namely, as workloads are executed, as resources are acquired and shared, conditions for optimality of a new, or even already running, workload might mutate. Thus, requiring the system to be flexible and to adapt to contextual changes.

Finally, the ability to explore and leverage local and remote information alike to make decisions about workload orchestration creates concerns about the privacy and confidentiality of this information. On one hand, a free flow of information would greatly benefit the ability of the system to make fully informed and optimal decisions, while the risk of leakage of sensitive, or even private, information would suggest a more restrictive approach. Part of the activities of WP4 explores how to leverage and expand state-of-the-art for the utilisation of privacy-preserving techniques within the decision-making process, would it be at model training or inference stage. Note that this activity is complementary to the work performed

© 2022-2025 FLUIDOS

Funded by Horizon Europe Framework Programme of the European Union

within the boundaries of WP5, where the focus of the research activity is the overall system security and privacy.

All these challenges are spawning from the interaction that WP4 is having with the other WPs, as illustrated in Figure 1. We want to remark how the requirements are primarily provided by use case providers, through the influence of WP7 to WP3 and WP2, which is then embraced in the activities of WP4 and WP5.

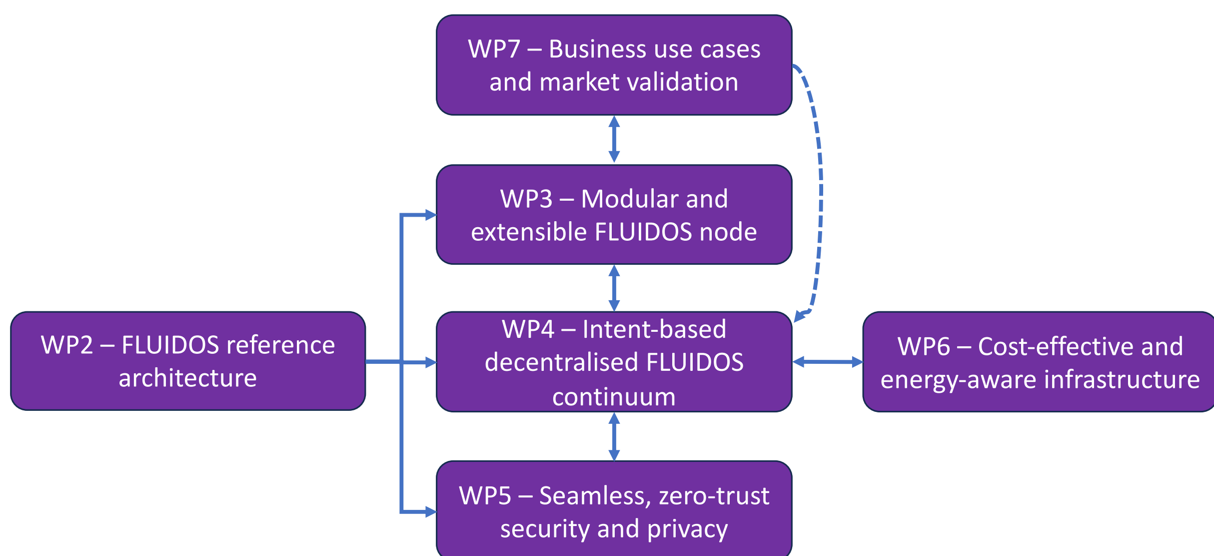
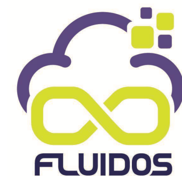


Figure 1 Relationship between technical WPs

In the remainder of this deliverable, we will present the preliminary results of the research activities conducted in WP4 to address the previously introduced research challenges.

Funded by Horizon Europe Framework Programme of the European Union



2 INTENT

The most generic definition of intent is the intention, purpose, or determination to do something. Within the cloud continuum provided by FLUIDOS, we use the term intent in a quite generic way. As reported in D2.1, we define intent as a way for the users to assign constraints to each workload execution through a high-level language without the need to be aware of the underlying computing infrastructure.

This definition, purposely vague, allows exploration of multiple approaches and techniques to both characterise and enforce user requests. Note that this generality of the definition requires some formal framework to allow users, and systems, to migrate between the possible semantic definitions and concrete serialisation. WP4 heavily collaborates with WP3 on this front, namely contributing and field testing the FLUIDOS ontology, which is presented in D3.1.

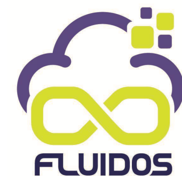
In the remainder of this section, we provide an overview of state-of-the-art and best practices for allowing users to express implicit and explicit requirements to workload execution.

2.1 MEDIUM-LEVEL SECURITY POLICY LANGUAGE

The Medium-level Security Policy Language (MSPL) [72] is a policy language originally defined to abstract the configuration languages with a vendor and control- independent format, which is organised by capabilities (e.g., a firewall from a particular vendor may implement packet filtering functionality in its capability set, while another vendor may implement also a deep packet inspection capability in his/her firewall). Unfortunately, defining this abstraction is not trivial because each security control has a specific syntax. Therefore, the mapping can be unmanageable in a generic syntax and MSPL is organised by security capabilities. A Capability is a basic feature offered by a security control (e.g., filtering, anti-spam, data protection, parental control). Therefore, MSPL is organised by a general model that defines the high-level concepts (policies, rules, conditions, actions, etc.) and a set of sub-models to capture the semantics specific concepts as attributes, condition types, methods, and more.

MSPL has been designed to satisfy the following requirements:

- Abstraction: the language must contain abstract security-related configurations, independent from a given vendor or product specific representation and storage. The reason for this requirement is that configuration semantics are independent from the actual representation. In fact, the same configuration settings can be represented and enforced in different security controls.



- Diversity: it must support the description of configurations for a variety of security functions (confidentiality, filtering, etc.). The configuration meta-model must furthermore support the configuration of such security capabilities, which follow different policies and concepts (e.g., communication protection, parental control), and are applied to different types of security controls.
- Flexibility and extensibility: it must be flexible and extensible enough to support the introduction of new security controls.
- Continuity: it must ensure the continuity of the policy chain, starting from HSPL down to the security control settings. This is useful for tracking which policy is enforced and which user is associated with it.

The MSPL language has been then extended [73] to further augment the expressivity of the language to additional domains, including the ability to characterise infrastructural and security requirements in IoT solutions.

Furthermore, in [74] MSPL has been extended, and leveraged, to allow reasoning on top of infrastructure and security requirements. This capability is being further expanded within FLUIDOS to support requirements beyond security. Thus, allowing the language, and associated solvers, to reason upon workload related user intents. Note that in this context, intents are required to be specified within the MSPL language. Examples of intent definition in MSPL are presented in Section 3.2.1.

Figure 2 depicts the relationships between the various projects and associated standards leveraged by FLUIDOS through MSPL.

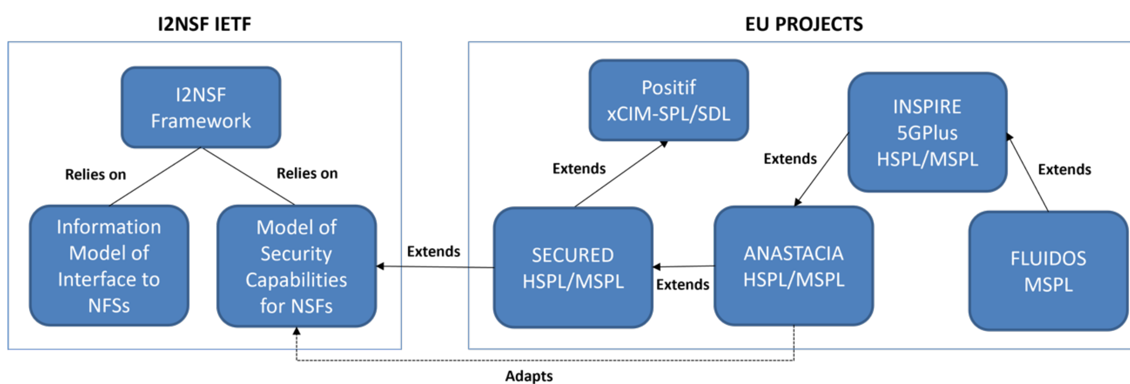
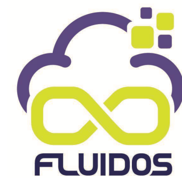


Figure 2 Relationship between FLUIDOS MSPL and prior work.

2.2 INTEL INTENT POLICY LANGUAGE

Funded by Horizon Europe Framework Programme of the European Union



An alternative approach taken by players like Intel [6], is to expand Kubernetes itself with Custom Resources (CRs) to allow the user to express intents in a declarative fashion. A user expresses their intents in form of objectives (e.g., as required latency, throughput, or reliability targets) and the orchestration stack itself determines what resources in the infrastructure are required to fulfil the objectives. This approach allows users to define an intent as a set of objectives related to a target Deployment, as shown in Figure 3.

```
apiVersion: "ido.intel.com/v1alpha1"
kind: Intent
metadata:
  name: my-function-intent
spec:
  targetRef:
    kind: "Deployment"
    name: "default/function-deployment"
  objectives:
    - name: my-function-p95compliance
      value: 4
      measuredBy: default/p95latency
    - name: my-function-availability
      value: 0.99
      measuredBy: default/availability
    - name: my-function-rps
      value: 0.0
      measuredBy: default/throughput
```

Figure 3 An example of intent manifest according to Intel's definition.

This relies on the ability to expand the supported objectives through external actuators for each supported objective.

2.3 TOPOLOGY AND ORCHESTRATION SPECIFICATION FOR CLOUD APPLICATIONS

The OASIS Topology and Orchestration Specification for Cloud Applications (TOSCA) [62] is the de-facto modelling and representation paradigm for specifying Infrastructure as Code (IaC). TOSCA allows for the expression of topologies of cloud-based applications, their components, and relationships, through node and relationship templates. These templates represent respectively the resources or components to be used in the deployment and the dependencies between the nodes [63]. TOSCA is an open standard designed to be cloud platform agnostic, unlike similar orchestration standards such as AWS CloudFormation or OpenStack Heat, which are tailored to their specific platforms. TOSCA-compliant orchestrators such as xOpera [64] allow for cloud application deployment to arbitrary IaaS. In [65], Tamburri et al. elaborate on the possibility of intent modelling using TOSCA templates, and [66] describes a proof-of-concept Intent-based TOSCA Cloud Management Platform that transforms high level intents, written in a natural-like language, into TOSCA YAML IaC definition files. For a simple example, Figure 4 shows the definition of a single server with both host and OS characteristics defined according to the TOSCA Simple YAML schema.

© 2022-2025 FLUIDOS

Funded by Horizon Europe Framework Programme of the European Union

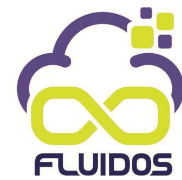
```
tosca_definitions_version: tosca_simple_yaml_1_3tosca_simple_yaml_1_3

description: Template for deploying a single server with predefined properties.

topology_template:
  node_templates:
    db_server:
      type: tosca.nodes.Compute
      capabilities:
        # Host container properties
        host:
         properties:
           num_cpus: 1
           disk_size: 10 GB
           mem_size: 4096 MB
        # Guest Operating System properties
        os:
          properties:
            # host Operating System image properties
            architecture: x86_64
            type: linux
            distribution: rhel
            version: 6.5
```

Figure 4 A simple "Hello World" for TOSCA.

2.4 NETWORK INTENT LANGUAGES

Similarly to TOSCA, a set of languages have been proposed to describe high level requirements for network infrastructure. From both the structural and performance point of view. This way, high-level network intents can be “translated” into low-level network configurations that can be pushed to configure network devices. Intent Description Languages (IDLs) such as the Network Intent LanguagE (NILE) [67], [68] have been introduced to help represent network requirements and constraints. In [67], the "Lumi" chatbot uses NILE as an abstraction layer between the natural language description of the network intent provided by the end user and the final Merlin [69] network configuration language. Additionally, [70] converts multiple network intents described using NILE into a P4 configuration file. Natural Language Processing (NLP) techniques are used in [67] and [71] to perform entity extraction, selection, and labelling, in conjunction with user feedback to correctly perform the translation of the intent into the intermediary representation.

Funded by Horizon Europe Framework Programme of the European Union



3 FLUIDOS NODE META-ORCHESTRATOR

This section provides an overview of the n scheduling, and meta-orchestration by the consortium members. It's important to note that this summary does not cover all ongoing or pending work at the time of this deliverable's preparation. Further details and developments will be included in the next report, Deliverable D4.2, and in regular periodic updates.

3.1 ALGORITHMS FOR ORCHESTRATION IN CLOUD TO EDGE CONTINUUM

This section describes the orchestration and scheduling algorithms proposed as part of the activities of WP4.

We focus on the details of each methodology, beginning with methods that are driving the orchestration according to specific purposes, namely an economical driven approach presented in Section 3.1.1, and another aiming at optimising network performance characteristics, namely latency, as presented in Section 3.1.2.

After that we focus on two main approaches for meta-orchestration. The first one relies on policy-driven meta orchestration, presented in Section 3.1.3, which allows users to specify higher level requirements for their application. On the other hand, Section 3.1.4 gives some insights on an AI-driven approach to meta-orchestration, where implicit and explicit requirements are used to identify optimal deployment characteristics.

3.1.1 Cost-based orchestration

To promote the vision of a fully decentralised virtual computing ecosystem, the FLUIDOS architecture enables a peer-to-peer sharing of resources between the different actors involved in computing environment creation. In practice, this is achieved by a dynamic peering protocol which oversees identifying the available peers and of negotiating the resources (e.g., CPU, RAM, Disk Memory) to deploy applications. The effect of this sharing strategy makes it possible for every service provider to access a pool of external resources when deploying their applications in addition to the ones already owned.

On the one hand, from a service provider perspective, the temporary acquisition of such resources (in other words, computational resources leased by other providers) can greatly increase the service quality performance. For example, it improves the application elasticity to absorb workload spikes during a heavy-loaded service requests scenario and enhances the cluster fault tolerance by distributing/replicating the microservices across different

© 2022-2025 FLUIDOS

Funded by Horizon Europe Framework Programme of the European Union

geographical regions. On the other hand, the leasing of resources from multiple infrastructure owners increases the economic expenditures sustained by each provider to run the related services which may severely hinder the advantages of the peering process.

Following this observation, the design of cost-aware orchestration algorithms covers an important role to mitigate the deployment costs by limiting the external resource consumption of the various applications. Although the cost minimization problem has been investigated on traditional cloud computing scenarios, which are essentially composed of physically co-located computing nodes, the FLUIDOS scenario entails additional challenges which require novel orchestration solutions. Unlike Li, Q. et al. in [14] and Samanta, A. et al. [15], who assume applications deployments on a restricted region, FLUIDOS nodes rely on computational resources that are geographically dispersed on a wide area, hence microservice-based applications might be split and executed at the edge or in the cloud, depending on their use case.

Moreover, existing works on resource allocation problems for the deployment of applications, such as Pallewatta, S. et al. [16] and Aryal, R.G.et al. [17], do not jointly consider their placement and communication requirements. Therefore, they are only partially effective in the FLUIDOS scenario, where the distributed application deployment imposes a fine-grained set of spatial constraints which in turn translates into latency and bandwidth requirements that various microservices need to satisfy to ensure the targeted quality of service. For example, some microservices in charge of the processing of real-time data must be at the edge for real-time applications. In contrast, batch workloads might require large volumes of computational resources only available on cloud nodes.

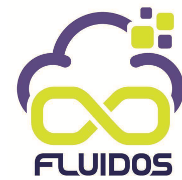
These heterogeneous deployment scenarios that, on one hand, prioritize applications deployment on few nodes to reduce the need of external resources, while one the other hand, promote the disaggregation of microservices across multiple nodes to better fulfil the latency and bandwidth service requirements, push for the necessity of an orchestration scheme striking a balanced deployment solution between both strategies. In this context, we investigate a cost-aware orchestration which quantifies the minimum amount of rented external resources needed by a service provider to ensure that the requirements posed by a given set of applications with locality constraints are accommodated.

3.1.1.1 System model

We consider a geographically distributed computing environment that spans both cloud and edge infrastructures. The physical part of the infrastructure is split in *D* domains. We model the infrastructure topology as a weighted graph:

$$G_D = (V_d, E_d)$$

Funded by Horizon Europe Framework Programme of the European Union



where V_d is the set of nodes corresponding to the various domains which have associated a cost, reflecting the cost to deploy a microservice, E_d is the set of edges representing the physical connection between domains, where E_i represents the links originating from domains i, and are characterized by some latency and bandwidth capabilities.

Each domain is owned by different service providers which rent/lend some resources from/to other providers through an automated brokerage system managing the whole acquisition/selling process to deploy their applications.

From a service provider perspective, the external resources, which are the ones leased by other providers, are costly, compared to the internal resources, which are the ones originally owned, hence their usage should be minimized. We assume that the cost per unit of external resource is *w*=1, whereas is *w*=0 for internal resources. The service provider objective is to deploy *A* applications on its own domain. Each application is modelled as a directed graph:

$$G_A = (V_a, E_a)$$

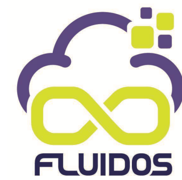
where the set of vertices V_a, represents the microservices composing the application, whereas the set of edges E_a, represents the communication flows between the various microservices.

Depending on the application demands, in some cases it may not be able to meet the applications' requirements based on its own infrastructure alone. This can be due to resource scarcity within the main domain or (e.g., not enough processing capacity is available to deploy all the applications) or because of locality constraints dictated by the distributed nature of the considered computing environment. Such constraints can be of two types:

- *Geographical*: an application requires some storage or processing in a specific geographical location, e.g., for privacy reasons. This is typically related to the elaboration of data gathered by specific sensors. A customary example is video streaming capture and elaboration from robots working within a factory facility.
- *Functional*: the main fog provider cannot secure a specific set of resources required by an application either because they are placed in a different domain or because the available resources are not sufficient for the application execution. For instance, an application may require a set of GPUs for fast processing, but the set of GPUs available to the main fog provider may satisfy such demand only partially.

In this scenario, the goal is to allocate the needed federated fog resources while satisfying any application's requirement and using self-owned resources as much as possible.

Funded by Horizon Europe Framework Programme of the European Union



3.1.1.2 Cost-aware orchestrator

We approach this problem by first formulating an optimization problem to compute the optimal application placement, then we approximate such a solution by designing a low-complexity heuristic.

Optimal solution

The deployment of applications is defined as a map that associates applications' microservices and their exchanged network traffic flows to the available fog resources. Formally the application deployment problem is formulated as an integer linear programming (ILP). In detail, we formalized the decision variables, objective function and the related problem constraint as follows:

- Objective Function: this is the cost of deploying a microservice onto a physical node of the infrastructure. The cost of such deployment is assumed larger when it is performed onto a node owned by other service provider, because their resources need to be rented. We assume a linear cost model, where cost of the rented external resource is directly proportional to the amount of bought CPU, RAM, and disk memory.
- Constraints: we define two sets of constraints. The first set takes care of the mathematical feasibility of the solution, meaning that we ensure that microservices are uniquely allocated on each node and that the output solution must be integer. Differently, the second set of constraints implements the deployment locality requirements by forcing a target subset of application microservices on specific domains to comply either to geographical or functional constraints previously defined. In addition, we ensure that communication flow between microservices also satisfies the required bandwidth and latency requirements and that these can be supported by the physical links connecting different domains of the infrastructure.

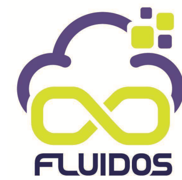
The proposed problem formulation can be essentially considered as a problem instance of computing the optimal topological mapping between virtual resources and physical resources represented as a graph. This representation corresponds to the well-known virtual embedding (VNE) problem, which is known NP-hard. For this reason, we also propose a heuristic algorithm to overcome the computation complexity of the optimal solution and make the proposed approach scalable for a larger number of nodes and applications.

Heuristic solution

The general idea adopted by most of the state-of-the-art heuristic solutions (e.g., [76]) is based on a Depth-First-Search (DFS) exploration of the search space for applications deployment. However, in our scenario, this may lead to situations where the resources of a

© 2022-2025 FLUIDOS

Funded by Horizon Europe
Framework Programme of the European Union



domain are saturated preventing the deployment of additional microservice whose locality constraints reside on that region. To overcome this limitation, we propose an alternative solution that is specifically tailored to the requirements of our problem, where locality constraints play a fundamental role, which is based instead on a Breadth-First Search (BFS) method. In particular, the key idea of the proposed scheme is the prioritization of the constraints during the partitioning phase to ensure that they are satisfied if there are enough resources left on the target domains. The algorithm consists of three main steps:

- Sorting of the batch of applications**:** all the applications are sorted based on their total bandwidth consumption.
- Partitioning of each application graph: following the application order defined in previous step, each application is partitioned based on the locality constraints defined on each microservice. This step defines the domain-based sorting of applications.
- Virtual node and link mapping: by iterating over all the applications in a breadth-first manner, this step explores all of them jointly and level by level. At each iteration, each microservices' subsets (as specified by the domain-based sorting) is mapped to a domain, where those ones with the lowest deployment cost are prioritized and, at the same time, the least-congested links (in terms of bandwidth) are used.

The total computational complexity of such algorithm is polynomial in the input size, which drastically reduces the execution time required to compute each applications deployment compared to the optimal solution.

3.1.1.3 Performance evaluation

Simulation setup

We assessed the results of the OPTimal and the heuristic solution, referred respectively as OPT and BFS, throughout simulations. We compared their performance against two state-of-the-art schemes leveraging a DFS approach named as:

- DFS_SoA_NoCost: this scheme does not consider the deployment cost optimisation; hence, its objective is just the maximisation of the number of deployed applications while trying to accommodate as many locality constraints as possible.
- DFS_SoA_Cost*:* this scheme minimises the deployment cost while completely ignoring locality constraints.

For all considered schemes, we analysed the cost performance, which essentially corresponds to the number of external and internal resources required to run the applications, and the feasibility performance which quantifies whether the computed deployment for each application batch can fulfil the services requirements. For each experiment we generated a batch of applications whose requirements in terms of CPU (measured in terms of Million

Funded by Horizon Europe Framework Programme of the European Union

Instructions Per Second, MIPS), storage, memory and throughput are uniform independent random variables with distribution values for each microservice as reported in Table 1. Unless stated differently, we assume that each batch is deployed on 3 different domains. Each data point in the reported graphs is the obtained average value over 30 randomised instances, where the network infrastructure does not change while the batch of applications and host distribution are randomly generated as described above. All the points are reported with their corresponding 95% confidence interval.

Number of domains D	Mean value	Range
CPU	1250 MIPS	[500,2000] MIPS
Memory	1.2 GB	[0.5, 2] GB
Storage	3.5 GB	[1, 8] GB
Throughput	3 Mbps	[1, 5] Mbps
Delay	262.5 ms	[25, 500] ms

Table 1 Applications' microservices requirements.

Cost and feasibility results

In Figure 5a, we can notice that OPT, BFS and DFS_SoA_NoCost have high feasibility. Specifically, the OPT and BFS strategies have a feasibility of 100%, meaning that they can deploy the complete batch of applications in all the 30 randomised instances (note that feasibility refers to the percentage of instances that are feasible). Additionally, looking at the deployment cost, which is computed as the sum of the costs associated to the used external resources, in Figure 5b we can see that BFS offers a solution very close to the optimal one (OPT).

© 2022-2025 FLUIDOS

Funded by Horizon Europe Framework Programme of the European Union

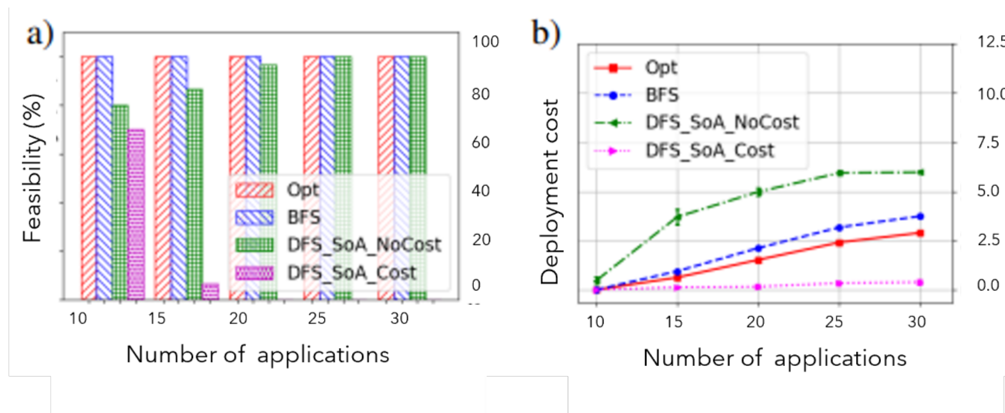


Figure 5 Feasibility-optimality trade-off. a) Feasibility percentage; b) Total deployment cost for each application.

DFS_SoA_NoCost, on the other hand, leads to a high deployment cost even though it has a good feasibility percentage. Conversely, in DFS_SoA_Cost the feasibility percentage is low as well as the deployment cost. This behaviour is reasonable given the greedy nature of the DFS approach. Indeed, if we include a cost optimization in such an approach, the algorithm prioritises all the regions with the lowest cost (that is, the regions in the main domain) for the deployment of all the applications' microservices. In this manner, resources with lower cost are quickly saturated precluding the possibility to satisfy the locality constraints for the applications that have not been deployed yet.

On the other side, if we do not consider cost optimisation, all the regions are treated in the same way, increasing the chance of having a feasible solution while increasing the deployment cost too. From this perspective, the BFS approach is beneficial since it does not evaluate the deployment of each application at the time, but it considers the deployment of a part of every application at each iteration. Thanks to this property, this method leads to a high feasibility percentage, since it helps guarantee locality constraints, and to a strong reduction of the deployment cost.

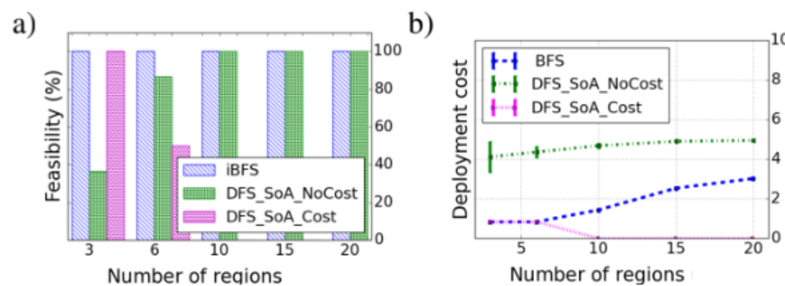
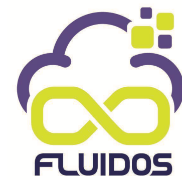


Figure 6 Realistic case with increasing number of domains. a) Feasibility percentage; b) Average deployment cost for each application.

Funded by Horizon Europe Framework Programme of the European Union



In Figure 6 we report the feasibility and the cost performance as the number of domains increases. Note that we omit the optimal solution due to the high computational time required to solve the optimisation problem. BFS achieves 100% of feasibility for all the sizes of the physical networks and presents the lowest deployment cost as the number of regions increases. Even when more resources are available with a higher number of regions, DFS_SoA_Cost has a low percentage of feasibility, thus highlighting the need to perform a breadth-first search in the state space for the deployment of applications presenting these locality constraints.

Scalability results

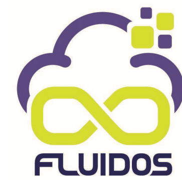
We analyse the scalability performance of the various schemes by measuring the execution time required to complete the deployment solution. Results are presented in Table 2 and Table 3. We remark that the values of OPT refer to executions that are stopped after 5 minutes if the solver has not completed the computation in that time range. To provide a comprehensive analysis, we assess the results by varying the number of deployed applications and then by varying the number of domains (in the latter case, we average the execution time performance across all deployed application batches). In general, the higher scalability of all heuristic approaches is apparent compared to OPT. Note that DFS_SoA_Cost has lower execution time than the other approaches because its execution is generally stopped earlier, i.e., when the algorithm cannot deploy one of the applications and the deployment is considered infeasible. Conversely, thanks to its polynomial time complexity, the proposed BFS scheme achieves comparable time performance while always ensuring the 100% feasibility of each application batch regardless of the number of available domains. Furthermore, the modest increase of the computational complexity makes it still suited for practical real-world scenarios.

Number of apps	OPT	BFS	DFS (SoA_NoCost)	DFS (SoA_Cost)
10	3.58	0.03	0.02	0.02
20	40.30	0.07	0.07	0.03
30	79.10	0.13	0.12	0.04

Table 2 Execution time (sec) with increasing number of applications to deploy.

Number of domains	BFS	DFS (SoA_NoCost)	DFS (SoA_Cost)

Funded by Horizon Europe Framework Programme of the European Union



3	0.14	0.09	0.09
10	0.22	0.10	0.10
20	0.50	0.31	0.33

Table 3 Execution time (sec) with increasing number of domains.

3.1.2 Latency-aware orchestration

With the growing significance of edge and fog computing paradigms in tackling concerns related to geographic proximity, minimised latency, and heightened privacy as highlighted in previous research [28][29], these approaches are progressively extending their scope to include smaller data centres located at the network's periphery. This expansion capitalises on shared foundational components to enhance service adaptability and creates a seamless transition from the edge to the cloud [30], particularly with the rising prominence of multi-regional cloud resources.

Many orchestrators have been developed in recent years, to optimise computational resources like CPU and Memory [31]. Nevertheless, the uniqueness of the edge-cloud continuum poses several challenges to traditional orchestrators. A critical observation is the absence of innate capabilities to dynamically adapt to variations in network metrics and the absence of real-time metrics in decisions. In addition, within the edge-cloud continuum, scheduling decisions can have a significant impact on the performance achieved by applications managed via Kubernetes.

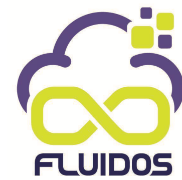
Recent solutions such as [32][33] introduce custom Kubernetes plugin for network-aware pod placement strategies for telemetry aware schedulers, or in [34] pods are offloaded in a three-tier network, blending cloud and edge computing to minimise the latency.

In contrast, we design a multi-cluster scheduler that goes beyond meeting user-specified intentions and places a strong emphasis on addressing user-perceived latency, but also makes optimal use of the cloud-edge continuum.

Our approach attempts to dynamically (de)allocate pods based on real-time latency measurements to satisfy users' needs. Our custom Kubernetes scheduler considers a variety of distributed peering clusters and arranges pods within the nearest node to minimise the latency perceived by users. The system is structured as a closed loop, constantly monitoring latency to adapt scheduling decisions and cater to user mobility requirements. We implemented our solution in an actual cluster system hosted at POLITO. We conducted a comparative analysis with the default Kubernetes Scheduler to showcase its efficiency in

© 2022-2025 FLUIDOS

Funded by Horizon Europe Framework Programme of the European Union



latency reduction. Furthermore, we assessed its intent-based functionalities and examined their influence on system performance and convergence time.

3.1.2.1 Latency-aware scheduler

The scheduling process provides a structure for assigning pods to computing nodes. Usually, it follows a request-driven methodology where it begins by filtering out nodes with inadequate resources and subsequently assigns the pod to one of the remaining nodes with sufficient capacity. This positions the scheduler as a crucial component responsible for resource management and determining workload adaptations. Moreover, it assumes a pivotal role in facilitating the execution of latency-sensitive microservices operating in the cloud-edge *continuum*.

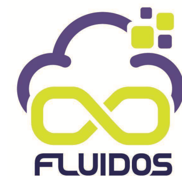
For latency-driven operations, we utilise two custom services: the ***Latency Meter*** and a specialised ***Latency Aware Scheduler***. The Latency Meter is installed on every worker node as a sentinel container within each pod replica. It serves as a proxy, intercepting each user request to gauge and record real-time network latency between users and nodes. When a request is received, it calculates the latency by comparing server and client timestamps and then stores this information in volatile memory for fast read-write access. The custom Latency Aware Scheduler can periodically retrieve these measurements.

This component integrates into the Kubernetes control plane, replacing the default scheduler and leveraging latency metrics for dynamic, informed pod scheduling. This custom scheduler comprises three key components:

- **Scheduler**: this component operates similarly to the default Kubernetes scheduler, but it incorporates a priority mechanism. The goal of this mechanism is to evenly distribute pods across all nodes, ensuring a diverse range of latency measurements.
- **Descheduler**: the descheduler periodically communicates with the Latency Meter to update the LatencyMeasurements (LM) data structure. Based on these measurements, it determines whether a pod should be reallocated and, if so, triggers the descheduling process. This dynamic interaction between the Descheduler and Scheduler determines which pod should be descheduled.
- **Latency Measurmentes (LM)**: This component stores latency measurements within an in-memory data structure. Although a persistent storage option could have been utilised, using volatile memory allows for quick data access.

When a user request is received, an exploration phase begins to collect measurements. Initially, no measurements are available, so pods are scheduled randomly. If the current configuration violates the latency constraints, the Descheduler acts by deallocating the affected pods. These pods are then reallocated to nodes that have not been explored previously to collect new latency measurements. A steady state is reached when the latency

Funded by Horizon Europe Framework Programme of the European Union



constraints are met. However, it's important to note that perceived latency can fluctuate due to dynamic network conditions or user mobility. Consequently, the entire process operates within a control loop that continuously monitors latency levels, leading to adaptive updates in scheduling decisions as necessary.

Our solution makes scheduling determinations in accordance with user-defined latency requirements, expressed as intent. Users can specify whether they have hard, soft, or no constraints through a YAML file that outlines the configuration of the Kubernetes cluster deployment. More precisely, within the YAML file, users can designate up to two values in the annotation: hard_constraint and soft_constraint. As we assume that latency variation within a cluster is minimal, the solution initially focuses on latency-aware choices to identify a suitable cluster. Subsequently, it employs load-balancing decisions to select the node.

When a hard constraint is set, it means that a stringent latency threshold must be strictly adhered to. In this scenario, if the exploration phase records measurements that surpass this stringent threshold, the Descheduler promptly releases the pertinent pods and requests their reassignment until the user-defined constraint is met. In this context, the system prioritises a swift and efficient mechanism over frequent pod migrations. The primary objective is to fulfil the specified latency constraint, representing a trade-off between minimising latency and ensuring operational stability. If the constraint cannot be met, the scheduler will opt for the cluster with the lowest latency and notify the user accordingly.

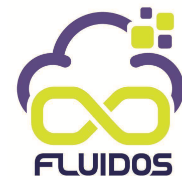
In the absence of a defined constraint, our scheduler aims to minimise user-perceived latency. To achieve this, it must gather measurements from all available clusters. Consequently, its core strategy is to distribute pod replicas across a range of nodes, while the Descheduler constantly monitors latency. Once the Descheduler accumulates measurements from all the application pod replicas, a loop mechanism initiates. This process involves removing pods from the cluster with the worst latency, effectively triggering the allocation of new pods on nodes that have not been explored yet. Fresh measurements are then collected, and the cluster with the highest latency is once again identified. After this series of local optimization steps, which can lead to suboptimal solutions, once enough measurements are gathered, the entire dataset is utilised to schedule pods on the clusters with the lowest global latency.

When a soft constraint is defined, the emphasis is on prioritising Soft Valid clusters, i.e., clusters whose latency stands between the hard and soft acceptable. Specifically, if we denote N_soft as the number of Soft Valid clusters currently hosting pods and N_hard as the number of Hard Valid clusters, i.e., satisfying the hard constraint, the Soft Condition is triggered when the following condition is met: $N_soft + N_hard > N_tot/ 2$, where N_tot represents the total number of clusters.

The Soft Condition mechanism is then activated, leading to the process of descheduling from Hard Valid clusters in favour of Soft Valid ones with the goal of further reducing latency.

© 2022-2025 FLUIDOS

Funded by Horizon Europe Framework Programme of the European Union



However, because this involves a greater number of measurements and reallocations and gives priority to Soft clusters over the others, meeting the soft constraint is achieved at the expense of load balancing and the time required for convergence.

Each time a new pod must be scheduled within a specific cluster, our scheduler employs a load-balancing strategy. To achieve an even distribution of pods across a range of nodes, a priority list is generated. A node's priority is determined by the number of pod replicas from the same application it accommodates: nodes with fewer pods are given higher priority. If multiple nodes have the same priority, the one with less resource usage, such as CPU and RAM, is selected. In the event of a tie, the choice is made randomly.

3.1.2.2 Performance Evaluation

To test this solution, we create 9 Kubernetes-driven clusters, where each worker node (2 per cluster) runs Ubuntu 20.04.6 LTS with 2 GB RAM, 2 CORE CPU and 20 GB disk, *containerd*[2] is the default container-runtime, and the Container Network Interface (CNI) is Flannel. The network interconnection among clusters is set up with Liqo[3] using the out-of-band peering, which includes initial authentication and communication with remote Kubernetes API servers, to flow independently of the VPN tunnel established between clusters. We then use the **tc** command to establish the latency range for each cluster. It's important to emphasise that there is an inherent baseline latency within the clusters, which we empirically determined to fall within the range of 15 to 25 milliseconds.

We have devised three distinct scenarios to investigate both the success rate and latency distribution in diverse network conditions:

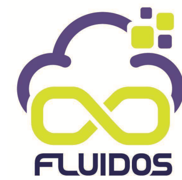
- *Scenario 1* enforces stringent latency constraints, which pose a significant challenge for many clusters to meet. This scenario serves as an example of a high-demand, low-tolerance application setup.
- *Scenario 2*, on the other hand, relaxes the latency limits and comprises clusters with low latency, reflecting a more accommodating network environment.
- *Scenario 3* is characterised by clusters with higher latency but still within acceptable latency bounds.

[2] https://containerd.io/

[3] https://liqo.io/

© 2022-2025 FLUIDOS

Funded by Horizon Europe Framework Programme of the European Union



This diversity in scenarios *reached by varying the latency via **tc**) allows us to assess the capability of schedulers in deploying pods in the most suitable clusters.

To gather latency data, we configure the latency meter to collect values every 5 seconds, and the descheduler operates every 30 seconds. This configuration aims to prevent frequent migrations and ensures a reliable evolution of latencies. In each scenario, we conducted 100 tests and reported the average values.

3.1.2.3 Convergence and Accuracy of the Solution

In this series of experiments, we conduct a comparative analysis between the Default Kubernetes Scheduler and three variants of our proposed solution:

- *Hard-compliant*, where our latency-aware scheduler focuses solely on meeting the hard constraint.
- *Soft-compliant*, where the scheduler also tries to fulfil the soft constraint.
- *Min-Lat*, where the scheduler prioritises selecting the node that minimises user latency.

These variants are tested in all three scenarios to verify our solution’s ability to satisfy latency requirements. Therefore, we begin by evaluating the schedulers' ability to meet the required latency. Figure 7 depicts the distribution of latency values (as a box plot) and the success rate for soft constraints (depicted as a blue curve) for all the tested schedulers across multiple scenarios. It becomes evident that the Default Scheduler demonstrates subpar performance in various scenarios, with an average latency of 70.52 ms, which is the highest among all the schedulers. Furthermore, the latency varies significantly among runs due to the latency-agnostic nature of pod scheduling by the Default Scheduler.

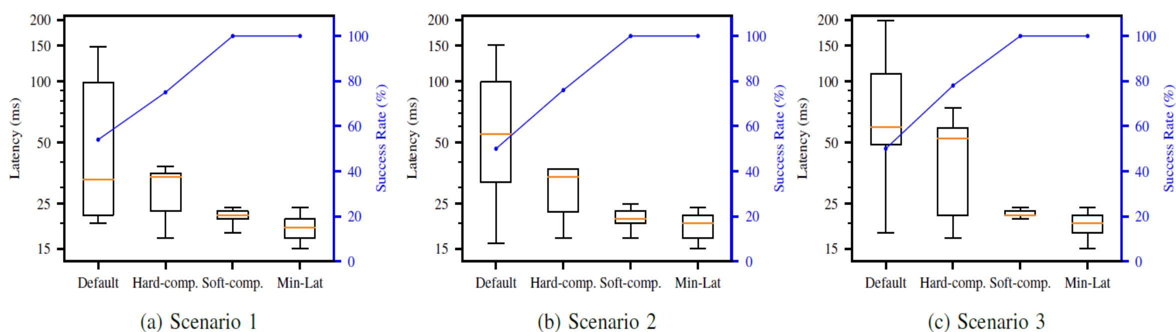
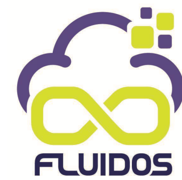


Figure 7 Latency (box plot) and success rate (blue curve) for all testing scenarios.

When evaluating the success rate, the Default Scheduler consistently achieves only around 50%, indicating that it essentially performs at the level of a random decision when placing a pod in a valid cluster. In contrast, Hard-compliant shows a substantial reduction in latency

Funded by Horizon Europe Framework Programme of the European Union



compared to the Default Scheduler in all scenarios, with a success rate of 76.5%. However, it doesn't reach 100% success because it prioritises "hard" constraints over "soft" ones. Both Soft-compliant and Min-Lat consistently maintain very low latency, typically below 22 ms, with Min-Lat slightly outperforming Soft-compliant in most scenarios. Both solutions achieve an impressive 100% success rate, demonstrating their high reliability in ensuring task execution. Notably, Min-Lat may require a longer convergence time, and the cumulative distribution function (CDF) of convergence time for our schedulers is shown in Figure 8.

We compare against a similar latency-aware scheduler, namely [77] referred to as Latency-Aware Kubernetes (LAK) in the following.

Comparatively, Hard-compliant consistently records the quickest convergence times, showcasing its rapid adaptability to system fluctuations. Soft-compliant takes longer to adapt but is still considerably faster than Min-Lat, which often exhibits prolonged convergence times, exceeding 241 seconds. In summary, the Default Scheduler lacks latency correction and places pods in clusters without any consideration of latency, resulting in random selections. On the other hand, the scheduler of our solution offers clear advantages.

LAK, instead, by not considering the union of clusters but treating them individually, takes much longer to find suitable nodes to host the applications in all three scenarios.

Hard-compliant strikes a balance between speed and performance, making it suitable for dynamic environments. In contrast, Soft-compliant and Min-Lat are optimised for high-performance solutions. Therefore, the choice of scheduler can be made based on specific requirements, whether it's adaptability or swift action.

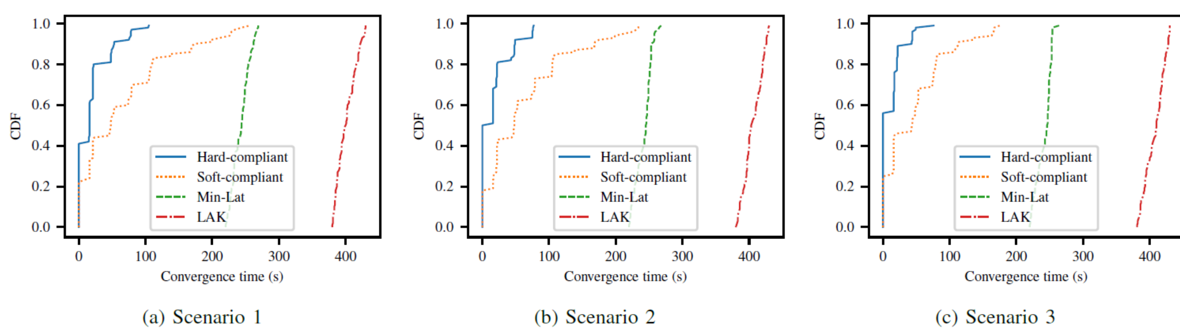


Figure 8 CDF of converge time for the three scenarios and comparison with alternative Latency-Aware Schedulers (LAK).

3.1.3 Policy-based Meta-orchestration

Funded by Horizon Europe Framework Programme of the European Union

Distributed computing is transforming the way services are provided, especially when solutions focus on collaboration and aggregation of resources provided by different entities or organizations. As a result, challenges arise such as heterogeneity of infrastructures and technologies, which has a direct impact on the management of these infrastructures [75]. To try to manage a distributed infrastructure when dealing with service deployment requests, we cannot only rely on an allocation/orchestration algorithm, this is where meta-orchestration arises, with the objective of choosing the best algorithm given the current situation of such a

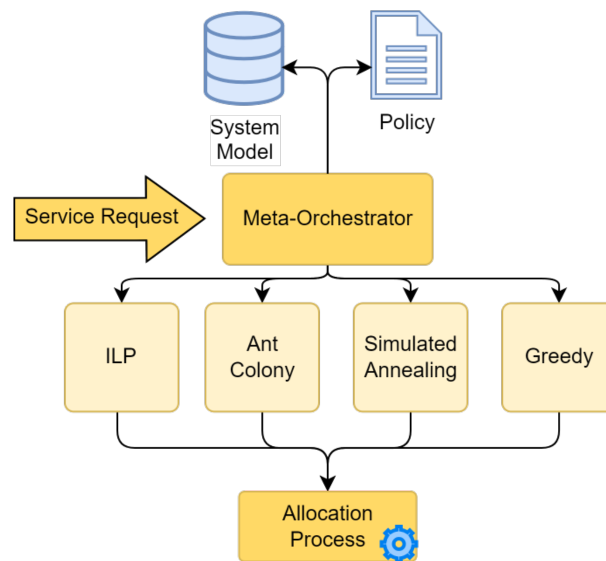


Figure 9 Meta-Orchestration Process

dynamic infrastructure as distributed environments and the policy received, once the orchestration/allocation algorithm is chosen, it will be in charge of solving the allocation placement problems which are those that once solved, it knows where the service is going to be deployed.

Funded by Horizon Europe Framework Programme of the European Union

The orchestrator is the brain of the architecture and in general will provide us with three fundamental answers: what, where and how. These answers are broken down along various stages of the orchestration process. The "what" refers to what will be deployed, the "where" means where it will be deployed and the "how" encompasses the strategy and methodology by which the service will be deployed. This aspect involves critical decisions, such as the efficient allocation of resources, network configuration, capacity management, etc. As we move through the various stages of orchestration, the orchestrator becomes the master architect, comprehensively coordinating the "what", "where" and "how" to achieve a smooth and efficient implementation of the required services. The meta-orchestrator's decision to determine which algorithm to choose is currently governed by the algorithm that offers the fastest decision time based on the number of nodes available in the infrastructure and the number of constraints that have been requested in the policy (both soft and hard). As shown in the Figure 9, there are four allocation algorithms, which have been designed following a common structure to provide a modular and adaptable approach which gives an essential flexibility for the incorporation of new algorithms and the continuous improvement of the system performance.

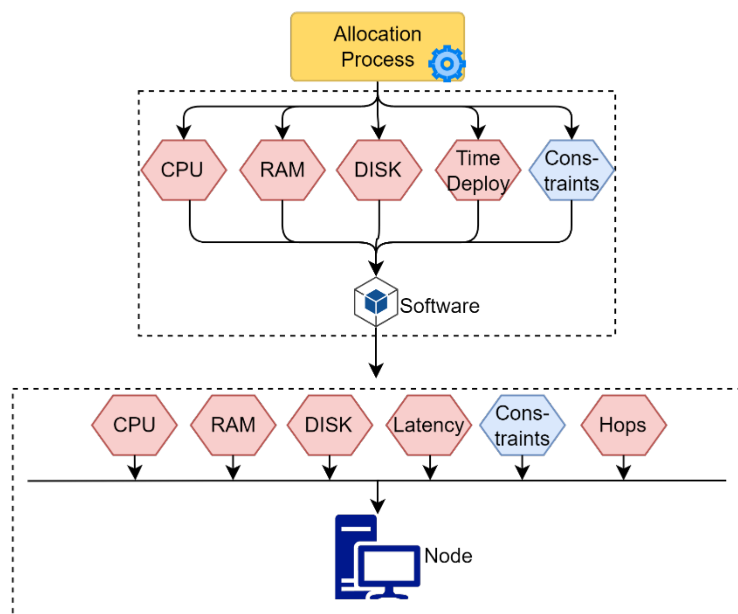
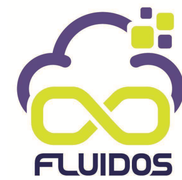


Figure 10 Allocation Process

The allocation process developed in scope of FLUIDOS has as goal provide what software is going to be deployed and where. For this purpose, as shown in Figure 10,it is divided into two phases:

In the first phase the software to be deployed is selected so the orchestrator provides a set of candidate software depending on capacities that come extracted from the policy which has been translated and the orchestrator has inferred. For example, a policy that asks to deploy

Funded by Horizon Europe Framework Programme of the European Union



a HTTP Server, the orchestrator knows that this capability belongs to Apache and Nginx software, so it proposes them. The algorithm, depending on the constraints, will select the software with the lowest computational cost for the enabler/node, so it considers software characteristics such as CPU, RAM, DISK consumption, time it takes to deploy, etc. Based on weights that can be dynamically assigned to these variables and the constraints determine the software to be chosen.

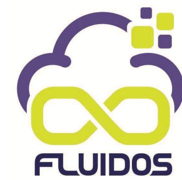
The last phase determines the node to be chosen by selecting from a set of candidate nodes. As in the first phase, these nodes are chosen based on their capabilities (for example, nodes with specific capabilities such as TEE, honeypots). In this phase, the selection of nodes/enablers aims to minimise the computational impact on the nodes, considering factors such as CPU, RAM, disk, and latency. Dynamic weights are applied to these factors to determine their importance. In addition, if constraints require it, values such as hops, service distance, bandwidth and location awareness can be considered. The latter implies that if the data does not have to pass through a particular location, e.g. France, the algorithm will exclude those nodes where the data would pass through France. For example, the policy could have a constraint of the type "the HTTP server has to be deployed in France with a latency of less than 30 ms", so the algorithm will try to select the most suitable node. Once the allocation algorithm is finished, it returns the software and the node to be deployed and passes it to the orchestrator to continue with the orchestration process and as the brain of the architecture it oversees executing and storing a new enabler in the system, continuing with the example, the new enabler would be the HTTP Server.

It is crucial to note that the work of the orchestrator goes beyond the simple deployment of services. The deployment of a service represents only one phase of the orchestration process, specifically called the allocation process. Orchestration encompasses a wider range of functions, some of which include translating policies into a format understandable to the system, checking for possible conflicts or dependencies in input policies and reactively resolving such problems, storing information regarding deployments, and invoking the various fluid components for monitoring, resource acquisition, etc. For example, consider the scenario in which two services are to be deployed, and one of them depends on the other. In this situation, the orchestrator would generate events to effectively address and resolve these problems. Similarly, if a resource shortfall is detected, the orchestrator would take on the task of contacting the relevant components responsible for resource acquisition, all for the purpose of deploying the service. Therefore, the orchestrator will provide FLUIDOS with the versatility to deal optimally and successfully with service deployments, service management and as management infrastructure.

3.1.4 Model based meta-orchestration

The orchestration task which is to determine the most optimal node (or FLUIDOS node) for a given deployment (set of constraints) which can be formulated as a bin backing problem

Funded by Horizon Europe Framework Programme of the European Union



(objects of different volumes must be packed into a finite number of bins or containers each of fixed given capacity in a way that minimises the number of bins used) is a NP-hard problem. As it was mentioned, mathematical modelling techniques attempt to solve the orchestration problem by means of ILP and then use standard techniques to find optimal solution for the problem However, due to their high computational cost, ILP formulations can only be used for small size problems. Moreover, there is no polynomial complexity algorithm to find optimal placement for large size problems. Therefore, most of the reported techniques are applying some heuristics to find approximate solutions to the problem.

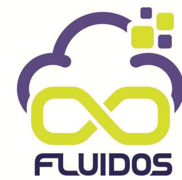
3.1.4.1 Background

The implementation of the orchestration engine as a set of constraints (rules) is presented, for instance, in [1] by Santos, J. et al. A network-aware framework for a K8s platform named Diktyo is developed to determine the placement of dependent microservices in long-running applications focused on reducing the application's end-to-end latency and guaranteeing bandwidth reservations. Guim, F. et al. in [2] present a solution for intelligent dynamic resources configuration on edge computing platforms hosting multi-tenant services while guaranteeing the Service Level Objectives (SLOs) for each service and helping green communication goals. The Multi-service Task Computing Offload Algorithm (MTCOA) is proposed in [3] by Song, S. et al. The algorithm is designed to solve the NP-hard Mobile Edge Computing (MEC) problem by minimising an objective function on the offloading cost of multi-service tasks to obtain the optimal unloading plan of the edge and remote cloud system.

At the same time, recent solutions expand the orchestration domain to a distributed setting. One of approaches which proposes an orchestration platform for multi-cluster applications on multiple geo-distributed Kubernetes clusters, mck8s, is presented in [4]. The platform, developed by Tamiru, M.A. et al. offers controllers that automatically place, scale, and burst multi-cluster applications across multiple geo-distributed K8s clusters. In the work [5] Gabriele, C. et al. present a DRAGON, the Distributed Resource Assignment and Orchestration algorithm that seeks optimal partitioning of shared resources between different applications running over a common edge infrastructure.

One of the key inputs for the orchestrator is a list of necessary resources which impacts a final deployment decision. In [6] Intel introduces Intent-Driven Orchestration, which allows to use *intent* instead of declaring a set of resources: a user expresses their intents in form of objectives (e.g., as required latency, throughput, or reliability targets) and the orchestration stack itself determines what resources in the infrastructure are required to fulfil the objectives. Following the similar path, Morichetta, A. et al. in [7] develop a methodology for deploying an intent-based system for the computing continuum and implement an architectural framework leveraging the serverless paradigm. The framework focuses on defining and

Funded by Horizon Europe Framework Programme of the European Union



implementing the main components for translating the management requirements into actions executed by serverless functions inspired by a three-layer model.

Wu, C. et al. in [8] propose an Intent-driven DaaS Management (IDM) framework to autonomously determine the cloud-resource-amount configurations for a given DaaS QoE (Desktop as a Service Quality of Experience) requirement. Metsch, Thijs et al. in [9] propose an intent-driven orchestration method to enable service owners to express the desired target key performance indicator (KPI) objectives for their service components instead of declaratively defining the required state and resources.

Nowadays *Machine Learning* methods are widely applied in various areas, including container orchestration. A comprehensive literature review of existing machine learning-based approaches was conducted by Zhong, Z. et al. in [10]. According to authors, ML-based models could produce more accurate orchestration decisions with shorter computation delays, under complex and dynamic cloud environments consisting of highly heterogeneous and geographically distributed computation resources, compared with traditional heuristic methods. For example, in [11] Han, Y. et al. introduce KaiS, a learning-based scheduling framework for K8s-oriented edge-cloud systems to improve a long-term throughput rate of request processing. KaiS adopts a two-time-scale scheduling mechanism to balance request dispatch and service orchestration.

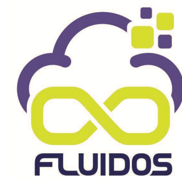
In [12] Lou, J. et al. explore Reinforcement Learning models being applied to task scheduling domain. The proposed approach jointly considers assignment and migration for mixed duration tasks and devises a novel energy-efficient task scheduling algorithm that outperforms the existing baselines in terms of energy consumption while keeping the same level of QoS. Iftikhar, S. et al. in [13] propose an approach called HunterPlus which examines the effect of extending the Gated Graph Convolutional Network's GRU (Gated Recurrent Unit) to a BGRU (Bidirectional Gated Recurrent Unit). Authors also study the utilisation of Convolutional Neural Networks (CNNs) in optimizing cloud-fog task scheduling. Experimental results show that the CNN scheduler outperforms the GGCN-based models in both energy consumption per task and job completion rate metrics.

3.1.4.2 Proposed approach

The intent can be *explicit* and *implicit*. Explicit intents are the actual requirements from the user/client which need to be satisfied for successful performance of the deployment.

The key component of any intent (represented by the pod manifest) is an image/list of images intended to deploy. To estimate the optimal placement for the image(s) the orchestrator needs to have an ability to evaluate the image in advance to deployment. A direct solution is

Funded by Horizon Europe Framework Programme of the European Union



to benchmark the input image as it appeared to the orchestrator to evaluate desired statistics which can influence requested intents. Despite the potential benefits (such as guaranteed realistic characteristics) this solution lacks generalization as not all types of the images can be benchmarked this way. Service applications such as "nginx" can demonstrate low resource demand on the benchmark but perform different depending on the actual use case. Also, such metrics as latency or throughput are highly varying through time which makes only benchmarking insufficient measure. Hence, a more advanced approach is needed. The model-based orchestrator trained on large datasets which contain history of deployments for the given intent (and the following performance) can learn (and update) the dependency between the intent and the successful deployment decision.

The idea behind the model-based orchestration proposed during the project is to formulate the intent-based node orchestration problem as a user-item engagement, i.e., utilize a recommendation system approach to narrow down the number of the suitable resource sets (FLUIDOS nodes). Modern recommendation algorithms are widely used in Big Data because of their ability consider both user and item features and infer dependency between user preferences and an item affinity to a user. Also, the important feature of recommendation algorithms is flexibility in updating recommendation as a new item is appearing in the system.

Therefore, to recommend the FLUIDOS node, characterized by the available resources set, for the user, represented in the input intent, we are exploring Machine Learning (ML) approximation of Matrix Factorization (MF) [27]. When formulate the orchestration problem by means of recommendation approach the explicit intent can be represented in a form of a pod manifest (written in YAML). To adjust the manifest to be used by the model it is transformed into high dimensional sentence embeddings. These embeddings are feed into a feedforward neural network. The FLUIDOS nodes configurations (available resources and additional features, e.g., location) are represented by variable-length sequence feature vectors which are mapped to a dense vector representation via a node embedding layer. The FLUIDOS node embeddings are learnt jointly with all other model parameters through normal gradient descent backpropagation updates. The model then learns to predict the optimal node configuration for a given explicit intent.

3.1.5 Orchestration on the Edge

Utilizing cloud resources in IoT applications suffers from two limitations: high latency and high network bandwidth consumption. This is commonly addressed by shifting processing to hierarchical, layered network processing, i.e., at the edge, fog, and cloud layers.

However, the new Edge computing solution can readily adapt to diverse sensor setups and settings thanks to the use of FLUIDOS Edge orchestrator. Instead of static deployments, the Edge orchestrator objective is to dynamically interact with the physical space and gather contextual information about the environment, which is used to select which sensors and

Funded by Horizon Europe Framework Programme of the European Union

analytics workflows to conduct. Figure 11 depicts the overall vision based on the FLUIDOS edge architecture as defined in D2.1. The orchestrator manages services and edge resources, based on application intended metrics, such as response and reaction latency, or achieving good quality results of a particular deployed application, while incurring low computational, networking, and energy costs. Additionally, the edge orchestrator can benefit the needs in continually changing physical environments, and can guarantee resource-efficiency (e.g., generating less data, hence consuming less resources) and respect battery-operated sensitive devices.

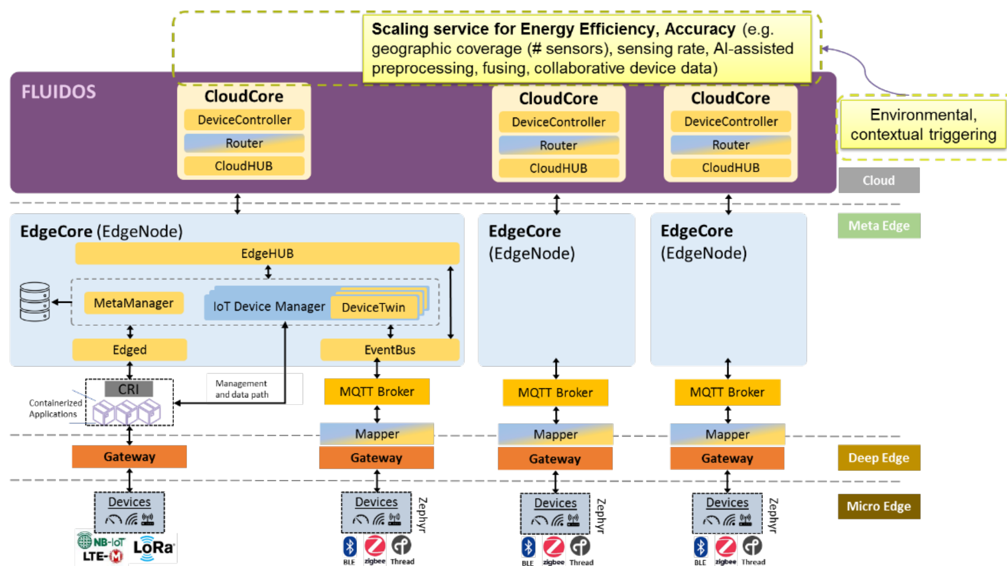


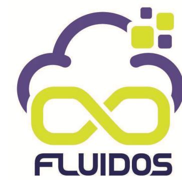
Figure 11 Orchestrating Edge Applications

With FLUIDOS, edge orchestration in a networked IoT infrastructure aims to create a more intelligent system by automatically handling real-time edge events, traffic, or other dynamic demands at the network's edge. This allows for more efficient resource deployment and near-instantaneous delivery of edge services. It can also assist in reallocating edge resources such as sensors or gateways with attached sensors among numerous edge devices.

Our research related to FLUIDOS edge orchestrator concentrates on the FLUIDOS scheduling extensions, which utilizes the K8S scheduler tool responsible for scheduling the pods across available worker nodes in the cluster. Pods represent a group of one or more containers inside which applications are running; it is the smallest schedulable component, i.e., each pod must in the same execution environment. Scheduling is usually based on the pods' resource constraints and requirements (e.g., Memory and CPU requirements), as well as the current resource utilization, most often independent of the total resources available. Hence, newly created pods, which form the smallest schedulable unit are scheduled on a suitable worker node with at least as many resources, usually with a fast best-fit algorithm. This algorithm performs two operations in sequence: filtering and scoring. Filtering constitutes a set of hard

© 2022-2025 FLUIDOS

Funded by Horizon Europe Framework Programme of the European Union



constraints that a node must meet to be able to run the pod, while scoring is performed against a set of soft criteria.

Worker nodes are responsible for deploying the pods. Besides the runtime environment, they also run different distributed services, such as *kubelet* for monitoring and ensuring that pods and their containers are healthy, and *kube-proxy* for both internal communications among worker nodes (via host subnetting) and external communications to the control plane (API server, scheduler, controller manager, and cluster store based on *etc*.).

Since the default Kubernetes scheduler is not aware of the topology of the worker nodes, it often results in non-optimal pod placements that violate hardware/software performance requirements. Hence, since 2019, hundreds of published papers have focused on extending the current state-of-the-art in Kubernetes scheduling [18][19][20]. Furthermore, different practical projects are supported by the Kubernetes SIG Scheduling Group [21]. SIG is responsible for organizing components that make smart Pod placement decisions. Some of these projects contribute to configuration options that control the scheduler: node selection via label or affinity, inter-pod affinity and anti-affinity specifications, and placement using taints/tolerations that provide limits for node sets and pods; these options could also be used to handle in a rather abstract way data locality issues, inter-workload interferences, and deadlines. In addition to configuration options, K8S scheduler extensions are provided. They extend from calling an external process via HTTP(S), to executing custom filtering and scoring methods, to enabling multiple plugins per pod that operate on the pod queue, to directly modifying the default scheduler's source code and executing custom schedulers (or related logic) per pod.

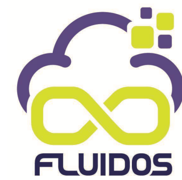
Recent extensions (or algorithms tested in simulators) refer to generic scheduling, multi-objective optimization, AI-based scheduling, and autoscaling-enabled scheduling for edge-cloud, fog, or cloud-based configurations [18]. Very few solutions refer to edge computing, where resources are severely limited, and different challenges apply. The research outlined below reflects the current state-of-the-art in Kubernetes scheduling related to edge computing, where constraints related to criticality and soft or hard real-time, performance, scalability, energy-efficiency, and reliability is very important.

Ogbuachi et al. used physical, operational, and network parameters and real-time monitoring information from node resources, such as load, temperature, and liveness, to improve fault tolerance and performance of dynamic orchestration in edge computing applications [22].

Haja et al. developed a custom, open-source Kubernetes scheduler extension that uses both periodic delay measurements across different edge nodes and edge reliability constraints [23]; this approach is suited for latency-sensitive edge applications. It is the only edge scheduler extension available in the open community.

© 2022-2025 FLUIDOS

Funded by Horizon Europe Framework Programme of the European Union



Wojciechowski et al. utilized dynamic network measurements gathered automatically by a dedicated infrastructure layer (implemented via the Istio Service Mesh) to improve inter-application node bandwidth and response characteristics [24]. This approach is crucial for the adoption of Kubernetes in 5G use-cases.

Oleghe used heuristic scheduling algorithms that use multi-objective optimization and graph network models concepts for container placement and migration in edge servers [25].

Li et al. improved dynamic scheduling by considering the disk I/O load of worker nodes [26]. Balanced-Disk-IO-Priority (BDI) focuses on disk I/O balance across nodes, while Balanced-CPU-Disk-IO-Priority (BCDI) considers CPU and disk I/O load imbalance on a single node.

Our contribution focus on improving FLUIDOS scheduling in edge computing ecosystems. Within this context, we develop innovative context-aware Kubernetes scheduling methods based on automated simulated annealing. Our methods leverage key system/network statistics with cutting-edge partitioning algorithms for dynamic workload management and resource allocation. Within this area, we examine dynamic adaptation to changing conditions using monitoring and data analytics, such as network routing statistics, while also considering computational resources, energy constraints and real-time response needs of the edge nodes. In addition, we plan to concentrate on evaluating the effect of operating system (GNU/Linux) kernel scheduler support when scheduling real-time processes with Kubernetes, focusing on low-level kernel load balancing and task migration.

Our open-source scheduler extensions will be evaluated using complex, dynamic workloads derived from realistic distributed embedded systems. More specifically, we plan to consider use-cases from IoT and Cyber-Physical Systems, related to Automotive, E-health, and robotics that involve delay-sensitive, bandwidth-intensive, power-efficiency, and reliability requirements. Different metrics such as performance, scalability, energy-efficiency, and reliability (e.g., computing node failures or restricted network connectivity due to network faults) will also be used to evaluate the efficiency of our scheduling extensions.

In summary, recent related work in scheduling edge computing clusters has focused on small configurations of 3 to 12 nodes (and almost always 3 to 4 nodes) while the work here described aims primarily at large scale deployments.

3.2 ARCHITECTURE

© 2022-2025 FLUIDOS

Funded by Horizon Europe Framework Programme of the European Union

We now describe the high-level architecture of the Service Handler and the Node Meta-Orchestrator components, as per the REAR protocol documentation[4]. These components, depicted in Figure 12, are the entry point for a user to interact with FLUIDOS.

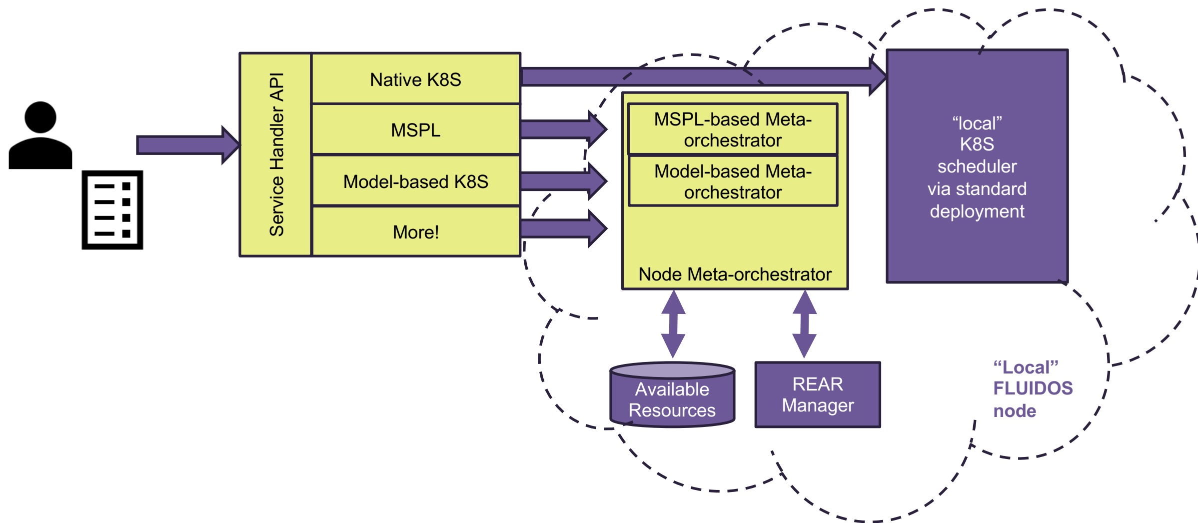


Figure 12 Meta-orchestrator conceptual architecture

Namely, the Service Handler receives a request directly from the user either from the provided kubectl plugin[5], or via direct API call. Upon receiving a request, the Service Handler inspects the payload to identify which meta-orchestrator is able to process such request.

Currently, FLUIDOS node meta-orchestrator operationally supports three main methodologies. The first one relies on traditional K8S orchestration, thus leveraging the off-the-shelves capabilities provided by Kubernetes. Thus, enforcing directly resource and node matcher information as provide by the user, and as available at the system level. The second methodology, presented in Section 3.2.1, leverages MSPL policy definition language, introduced in Section 2.1. Finally, Section 3.2.2 will present the model-based approach which expand traditional Kubernetes capabilities by inferring workload requirements based on priors, and explicit, user-defined, intent requirements.

3.2.1 MSPL-based Meta-Orchestration

[4] https://github.com/fluidos-project/Docs/

[5] https://github.com/fluidos-project/kubectl-fluidos-plugin

© 2022-2025 FLUIDOS

Funded by Horizon Europe Framework Programme of the European Union

To illustrate the behaviour of the MSPL-based orchestration algorithm we have the following example (Figure 13), consisting of two Kubernetes clusters (SuperNode) located in Spain and Italy respectively, both connected to each other through Liqo. Despite each SuperNode would contain all the FLUIDOS modules, for the sake of comprehension we will only provide a simplified version, providing a subset of FLUIDOS modules in the first SuperNode. Thus, the figure includes the user that will trigger the policies, the Service Handler that will handle the request, the Node Orchestrator which will provide the meta-orchestration capabilities, as well as databases such as the ratings and metrics and resources available that will provide relevant information to the node orchestrator during the orchestration process. The second node is shown to illustrate the offloading step, so no modules have been represented on it.

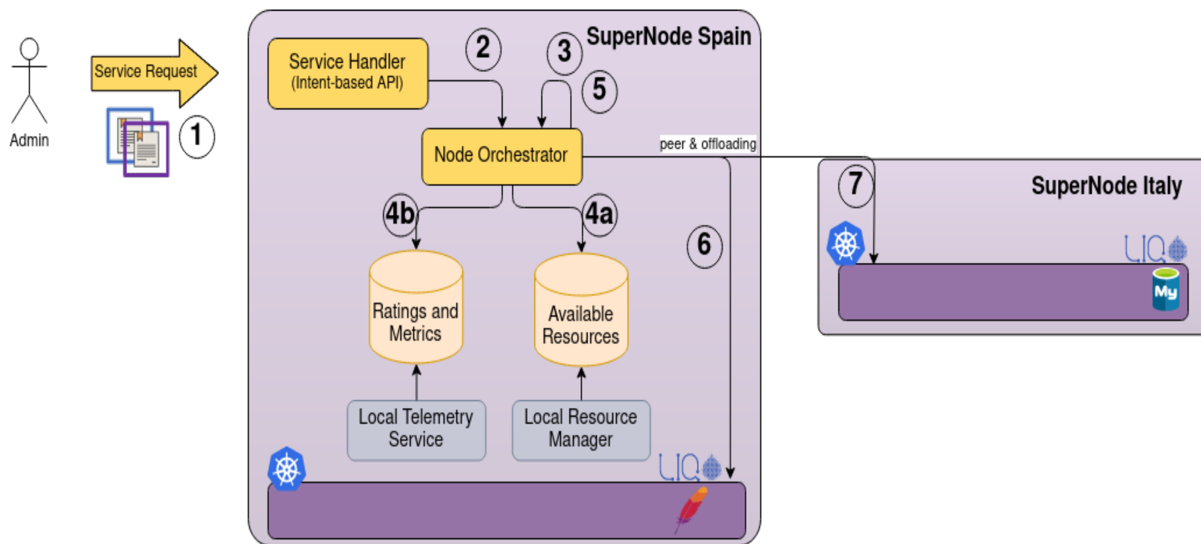
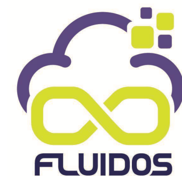


Figure 13 Example of MSPL-based orchestration algorithm.

Regarding the workflow, we will illustrate an example in which the user provides a request that cannot be fully accomplished locally as the request contains hard constraints regarding the location of the deployment, so it requires specific resources from a specific geography location. This approach works as follows:

1. The user wants to deploy two services (web and database) with certain requirements, so he models an MSPL policy that represents services and the associated constraints.
2. The Service handler analyses the policy/intent type. As the policy was modelled as MSPL policy, the Service Handler realizes that the policy must be forwarded to a Node orchestrator that instantiates MSPL-based orchestration capabilities.
3. The Node Orchestrator will decide to address it using the ILP-based solution so will model the problem as an ILP and solve it considering the use of RAM, CPU, and the defined constraints (soft and hard).

Funded by Horizon Europe Framework Programme of the European Union



4. During the orchestration step (3), information regarding available resources, software, metrics, and information regarding the underlying architecture are required.
5. Then a two-phase process will start, selecting in the first phase the best candidate software to use to deploy the service (e.g., Apache http, nginx, etc.) and finally determining which node is the most suitable to host the software.
6. In the case of our example, for the HTTP server intent we would first select for example NGINX as software and then a local node of the cluster as host. For the database intent, in the first step we would select MySQL, and when choosing the host node, the algorithm would fail because there is no local node that meets the requirement of being in Italy.
7. However, the algorithm would detect that the system has a fluid node (SuperNode Italy) that has a valid candidate, so it would perform a peering process and acquire the necessary resources thereby selecting that node as the final host.

Regarding the policies provided by the user, Figure 14 shows an example of MSPL orchestration policy composed of two different policies. As MSPL were extended in FLUIDOS for also providing services and constraints, the models use a new capability "Service MANO" that allows describing services and the associated constraints. In this example, the first policy requires the deployment of a service of type "HTTP_SERVER", and the constraints specify that the constraint type LATENCY must be less than 15ms. The second policy requires the deployment of a service of type "DATABASE" and it also specify as main constraint that the service must be geographically deployed in Italy.

```xml
<!-- Deploy HTTP SERVER with less than 15ms of latency -->
<?xml version='1.0' encoding='UTF-8' standalone='yes'?>
<ITResourceOrchestration
id="mspl_9f1a88b4fc67421b98de270d5a63d35f">
  <ITResource id="mspl_eef61525d1594412bdcef34a4bfb7fc8"
orchestrationID="mspl_9f1a88b4fc67421b98de270d5a63d34f"
tenantID="1">
    <capability>
      <Name>Secured_Service_MANO</Name>
    </capability>
    <securedService>
      <service id="132">
        <name>front_end</name>
        <type>HTTP_SERVER</type>
      </service>
    </securedService>
    <orchestrationRequirements>
      <hardConstraint>
        <constraintType>LATENCY</constraintType>
        <operator>LESS</operator>
        <value>15</value>
        <valueUnit>MILLISECONDS</valueUnit>
      </hardConstraint>
    </orchestrationRequirements>
  </ITResource>
```

```xml
<!-- Deploy Database in Italy -->
<ITResource id="mspl_eef61525d1594412bdcef34a4bfb7fc9"
orchestrationID="mspl_9f1a88b4fc67421b98de270d5a63d34f"
tenantID="1">
  <capability>
    <Name>Secured_Service_MANO</Name>
  </capability>
    <securedService>
    <service id="132">
      <name>data</name>
      <type>DATABASE</type>
    </service>
    </securedService>
  <orchestrationRequirements>
  <hardConstraint>
    <constraintType>LOCATION</constraintType>
    <operator>EQUAL</operator>
    <value>ITALY</value>
  </hardConstraint>
  </orchestrationRequirements>
</ITResource>
</ITResourceOrchestration>
```

Figure 14 Example of an MSPL policy.

It is important to highlight that despite in the example provided the second service is deployed using external resources as the constraint is based on the location, different kind of

Funded by Horizon Europe Framework Programme of the European Union

constraints are considered (e.g., CPU, RAM, Disk, Latency…). This means, the first service could be also deployed in the remote node if for instance, the local node is congested and the remote node is able to fulfil the latency constraint.

3.2.2 Model-based Meta-Orchestration

From an architectural point of view, FLUIDOS' model-based meta-orchestrator interacts with the local Kubernetes cluster through the operator paradigm, as presented in Figure 15. Namely, a Kubernetes Operator is a software component that reacts to events generated within Kubernetes as Resources are created, deleted, and updated. Specifically, we defined a new Custom Resource named `ModelBasedDeployment` to represent deployment requests defined through intents. Note that in this case the term "deployment" refers to the request of a user to deploy a given workload within the continuum.

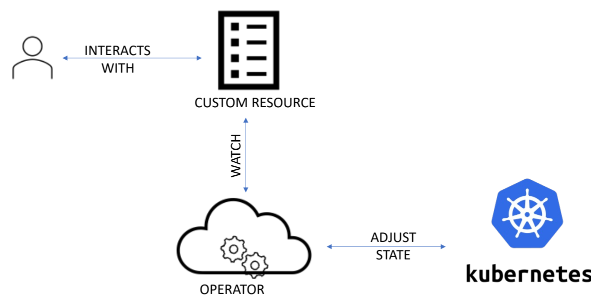


Figure 15 High level K8S Operator Paradigm

Namely, once a user request to deploy a workload, either expressed as Deployment or Pod, in FLUIDOS through the provide command line tool[6], this CLI [7] will wrap the request within a resource of type `ModelBasedDeployment`. This will cause the invocation of the Operator API of our custom operator.

Internally, the operator inspects and processes the request to identify, and expand, the intent defined within the request. Subsequently, the operator interacts with the ML component, which has been briefly introduced in Section 3.2.2. From an operational point of view, this component translates the workload information, including container image information, and explicit and implicit intents to a feature vector that is in turn used to perform a prediction with

[6]https://github.com/fluidos-project/Docs/tree/main/Work%20Packages/WP4#bonus-cli-extension

[7] https://github.com/fluidos-project/kubectl-fluidos-plugin/

Funded by Horizon Europe Framework Programme of the European Union

the provided model. The first version of the model can process explicit intents, such as CPU, Memory, Latency, Throughput and Location, as shown in the example of Figure 17.

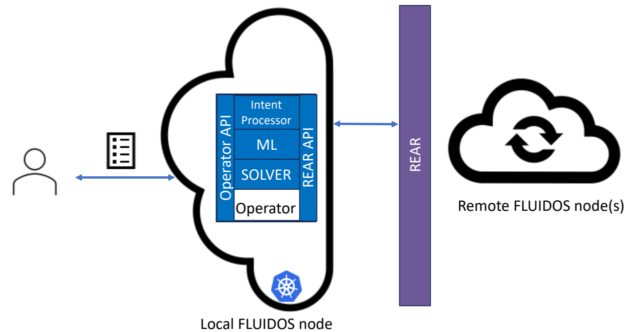


Figure 16 High Level FLUIDOS Model-based Meta-Orchestrator

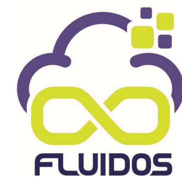
Future updates will focus on the integration of pod image(s) fingerprints and real-time node resources usage information to the model to further fine-tune the placement prediction.

```
apiVersion: fluidos.eu/v1
kind: ModelBasedDeployment
metadata:
  name: nginx-w-intent
spec:
  apiVersion: v1
  kind: Pod
  metadata:
    name: nginx-w-intent
    annotations:
      fluidos-intent-location: milan
  spec:
    containers:
    - name: nginx
      image: nginx:1.14.2
      ports:
      - containerPort: 80
      resources:
        requests:
          memory: "64Mi"
          cpu: "250m"
        limits:
          memory: "128Mi"
          cpu: "500m"
```

Figure 17 Example of pod manifest with explicit intent

The model then returns a template representing the resources required to execute the provided workload in such a manner that the associated user intent is satisfied. These

Funded by Horizon Europe Framework Programme of the European Union



resources include hardware characteristics, such as memory and processor values, the need to have access toa a GPU, computational and service characteristics, such as access to certain API servers, or database, and further meta-information including location, compliance guarantees, and security standards that the receiving nodes should provide.

The operator users this template to identify, with the Solver component, suitable resources locally and remotely. This is done by interacting with the REAR API, which allows exploring the resources offered, through the REAR protocol[8], by the participants of the continuum.

Material access to the remote resources is again done via interaction with the REAR API, which in turn leverages Liqo.

The current version of FLUIDOS model-based-meta-orchestrator is available as open source[9]. It is implemented in Python taking advantage of both native Kubernetes python API[10], and the Kubernetes Operator Framework[11].

[8] https://github.com/fluidos-project/REAR

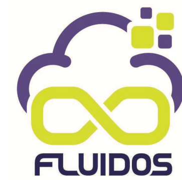
[9] https://github.com/fluidos-project/fluidos-modelbased-metaorchestrator

[10] https://github.com/kubernetes-client/python

[11] https://kopf.readthedocs.io/en/stable/

© 2022-2025 FLUIDOS

Funded by Horizon Europe Framework Programme of the European Union



4 PRIVACY & SECURITY OF MODEL-BASED INTENT-DRIVEN META-ORCHESTRATION

As indicated in the previous sections, AI-based algorithms for resource or workload allocation are a necessity, especially given the complexity of a dynamic, fluid, environment.

This scenario raises two main challenges. First, there is a concern with respect to how models, and models' predictions might leak private and sensitive information. This requires the models to be trained borrowing techniques from the privacy domain. Thus, we explore existing and novel AI-based approaches used for orchestration to adapt them to the FLUIDOS framework and add privacy constraints in the algorithm design/training. The second challenge derives from the ever increasing demand for data to train complex AI models. The established approach to the second challenge is to use a collaborative approach. Namely to share, perhaps publicly, large volumes of data to allow creation of more powerful and performant AI-based services. In an industrial scenario this would be a major concern, primarily because an internal dataset can potentially leak sensitive information, even when apparently anonymized. Moreover, traditional anonymization techniques are focused on protecting privacy of individuals, or of sensitive data-characteristics in this case. A well-established technique to collaboratively train complex models without releasing dataset is Federated Learning (FL). We plan to leverage techniques such as FL where FLUIDOS nodes collaboratively train AI model(s) without exchanging or collecting data, or differential privacy techniques where noise is added to data samples to preserve privacy to cite a few examples.

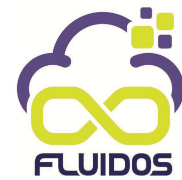
In the context of these challenges, we investigate how privacy-preserving machine learning (PPML) algorithms can be used to achieve two things at the same time:

1. facilitate FLUIDOS architecture with fundamental functionalities, and
2. protect the data, models, systems, and networks involved in the construction of said models within FLUIDOS infrastructure, from various adversarial attacks.

This exploration is motivated by recent findings in both academic and industrial research, which have shown that adversaries can execute a range of attacks aimed at extracting sensitive information from the parameters of the ML model and the data it utilises. Notably, these attacks encompass inference attacks, data reconstruction, and poison attacks. In response to these threats, researchers have introduced several countermeasures, falling into three primary categories: (i) homomorphic encryption, (ii) multi-party computation (MPC), and (iii) differential privacy (DP). However, these countermeasures are not without their drawbacks, as they often grapple with privacy, performance, utility, and fairness limitations. Also, homomorphic encryption, while practical, is constrained by its support for only a limited set of arithmetic operations in the encrypted domain. Fully homomorphic encryption, which allows arbitrary

© 2022-2025 FLUIDOS

Funded by Horizon Europe
Framework Programme of the European Union



operations, carries a significant computational overhead that hampers scalability. Similarly, solutions based on multi-party computation introduce substantial computational burdens and scalability limitations. Moreover, in some instances, differential privacy may fall short of delivering sufficient privacy when incorrectly applied or in cases of misguided trust assumptions, such as local DP versus central DP. Furthermore, if not harmoniously integrated with other critical aspects of the ML model, such as utility and fairness, it may adversely affect these dimensions.

In the next paragraphs, we cover the basic concepts that are used in our work, such as DP, FL, hierarchical FL, asynchronous FL, and combinations of them to achieve the objectives. These combinations are employed to train useful and privacy-preserving ML models for these functions within the FLUIDOS architecture. Towards this goal, we review important literature for these concepts. Then, we introduce our two novel approaches to train AI models in (a) hierarchical FL fashion, (b) asynchronous FL fashion, both designed to protect the model from adversarial attacks using differential privacy.

4.1 BACKGROUND

4.1.1 Differential Privacy

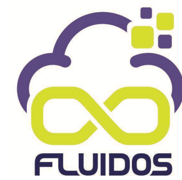
Differential Privacy (DP), introduced by Dwork et al. [35], is a foundational concept in privacy-preserving data analysis. To define DP, let **ε** be a positive real number, and consider a randomized algorithm A that takes a dataset as its input. An algorithm A is said to provide **ε**-DP if, for all datasets D1 and D2 that differ by a single element, and for all subsets O of the outcomes generated by running A:

$$\Pr[\mathcal{A}(D_1) \in \mathbf{O}] \leq e^{\varepsilon} \cdot \Pr[\mathcal{A}(D_2) \in \mathbf{O}]$$

This definition holds with the probability taken over the randomness of algorithm A. **ε**, often referred to as the privacy budget, quantifies the level of privacy protection provided. DP is achieved by adding noise selected from a specific distribution, which ensures the indistinguishability guarantee previously mentioned.

The traditional model of DP, known as the central DP (CDP) model, implicitly assumes the existence of a trusted entity that faithfully follows the protocol and introduces calibrated noise to guarantee DP. Consequently, the outcomes become differentially private through post-processing. However, in various real-world scenarios, these strong trust assumptions may not hold. To address this, the local DP (LDP) model assumes that each data contributor locally adds DP noise, i.e., in-situ, making sure that their inputs themselves are DP. Thus, any function

Funded by Horizon Europe Framework Programme of the European Union



computed on top of these DP inputs becomes DP through post-processing. Nevertheless, this approach has limited knowledge of the overall function being computed across the data and tends to overestimate the amount of noise required for privacy, therefore leading to loss of utility of the model trained.

The relationship between LDP and CDP hinges on the mechanism used to achieve DP. For example, the Laplacian mechanism ensures that **ε**-LDP also provides **ε**-CDP, illustrating the connection between these two approaches.

In this task, we leverage the principles of DP and its various iterations to craft efficient and effective PPML algorithms that can be used within the FLUIDOS infrastructure during e.g., resource orchestration.

4.1.2 Federated Learning

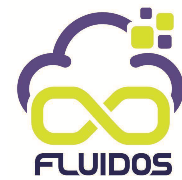
Federated Learning (FL) is a prominent option for deploying privacy-preserving ML [35]. FL offers a multitude of advantages and aligns with the pursuit of an ideal PPML algorithm, one that can train machine learning models beneficial to both centralised and distributed parties while avoiding exposure to unauthorised entities. In FL, the essential data required for ML model training is not directly transmitted between the parties, comprising clients and a central server. Instead, the process involves the distribution of model training across individual devices (clients) that possess localised data shards for training their respective local models. The global server subsequently collects and aggregates these local models to create a unified global FL model.

When compared to centralised approaches, FL presents a larger attack surface, primarily due to the following factors:

- Data distribution: The dispersion of training data across multiple devices heightens the risk of data breaches and attacks on individual devices with varying security levels.
- Communication channels: The communication between the central server and participating devices occurs over potentially insecure networks, rendering it susceptible to interception, eavesdropping, or man-in-the-middle attacks.
- Client-side vulnerabilities: Vulnerabilities within client or edge devices can be exploited, potentially compromising the integrity of the training process.
- Aggregation and model updates: The central server's role in aggregating model updates creates a potential vulnerability, with malicious clients having the capability to manipulate or poison updates, thereby compromising the integrity and fairness of global models.
- Privacy concerns: While FL aims to enhance data privacy by circumventing the necessity to share raw data for model training, it is not immune to attacks that seek to undermine the privacy of training data.

© 2022-2025 FLUIDOS

Funded by Horizon Europe Framework Programme of the European Union



To mitigate these risks and reduce the overall attack surface in the adopting infrastructure (e.g., FLUIDOS), it is imperative to implement robust privacy and security measures. These encompass secure communication protocols, encryption techniques, device authentication mechanisms, access controls, PPML methods, fairness assurance procedures, and the regular conduct of security audits.

4.1.3 Federated learning with Differential Privacy

In the pursuit of privacy-preserving machine learning executed in a decentralised or federated manner such as in the case of FLUIDOS architecture, several notable advancements have been made in the integration of DP with FL. McMahan et al. [36] were among the first to propose an approach that combined DP and FL, offering formal privacy guarantees. Similarly, Geyer et al. [38] put forward ideas in this domain. However, in both settings, the FL central aggregator had access to either noise-free gradients or noisy gradients from the participating clients, raising privacy concerns. To address this issue, Bonawitz et al. [39][40] introduced the concept of secure aggregation, which is a variant of MPC. Secure aggregation provides the central aggregator with an aggregated perspective of all gradients, irrespective of whether they are noisy or noise-free, effectively breaking the connection that could compromise privacy.

In a further development, Truex et al. [41] harnessed advancements from both LDP and MPC, employing threshold cryptography schemes to create a hybrid approach that offers enhanced privacy protection. Truex and colleagues [42] also proposed an approach that utilises LDP, albeit based on an alternative DP definition.

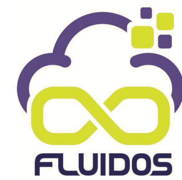
In this task, we harness synergies between FL and DP technologies, alongside the various innovative variants that have emerged, to craft privacy-preserving machine learning algorithms that are not only effective but also efficient, all within the framework of decentralised or federated execution.

4.1.4 Hierarchical Federated Learning

Abad et al. [43] have introduced a mechanism designed to facilitate communication-efficient and coordinated learning within the framework of Hierarchical Federated Learning (HFL). In this context, the concept of hierarchies is derived from the presence of clients communicating with smaller base stations or cellular towers, acting as intermediaries. These intermediaries, in turn, establish communication with macro base stations or the central aggregator. This hierarchical structure optimizes the learning process by efficiently managing communication flows.

© 2022-2025 FLUIDOS

Funded by Horizon Europe Framework Programme of the European Union



Similarly, Yuan et al. [44] have put forth a novel protocol aimed at enhancing communication efficiency in the Local Area Network-Wide Area Network (LAN-WAN) setting. This form of HFL differs significantly from the approach advocated by Briggs et al. [45], which focuses on segregating clusters of similar clients that can be independently trained using heterogeneous models. In all these prior works, a common thread is maintained: federated clients, much like the standard approach, interact with intermediary entities, which, in the case of Abad et al. [43], are the base stations. These intermediaries play a crucial role in consolidating information, typically on behalf of a subset of clients, to be communicated to the central aggregator(s), thus facilitating collaborative and efficient learning.

Therefore, we make use of the implicit and explicit hierarchies that exist in nowadays networks (including the one envisioned by the resources of FLUIDOS infrastructure), as well as the technologies of FL with DP and their different variants, to design scalable, hierarchical-based, privacy-preserving ML algorithms that are executed in a decentralized/federated fashion.

4.1.5 Asynchronous Federated Learning

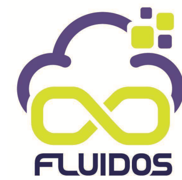
In the traditional FL paradigm, FL clients coordinate (synchronize) their training process. More precisely, they train local models on their datasets and periodically synchronize them with a central aggregator server. This synchronization, initiated and orchestrated by the server, ensures alignment among clients and collective updates of the global model in each round. There often are two main challenges in synchronous FL: scalability and dealing with stragglers.

Scalability is a critical concern, particularly in large-scale settings with many clients, as only a proportion of them may be accessible at any given time for training. Managing the training process across many clients can thus prompt significant challenges, encompassing tasks such as efficient communication, minimizing network congestion, and optimizing computational resources to accommodate the scale of the FL system. Additionally, the presence of stragglers, i.e., clients whose training progress is considerably slower than others, causes delays in the synchronization process and further complicates the learning process. To overcome these issues, the notion of Asynchronous FL (AFL) has been proposed as a variant to the FL framework where the communication between the clients and the central server is not coordinated in real-time [46]. The clients are not required to wait for other clients to complete their training rounds before sending their updates to the server. Rather, they can initiate communication with the server as soon as they complete their local training. This allows for more flexibility in the training process, as clients can proceed at their own pace and contribute updates whenever they are ready.

While AFL addresses some challenges, it also introduces new ones, including dealing with the heterogeneity of client devices, outdated updates, communication bottlenecks, etc. Nonetheless, various techniques in AFL can be used to ensure an effective training process

© 2022-2025 FLUIDOS

Funded by Horizon Europe Framework Programme of the European Union



[47][48][49][50][51]. In this work and task, we focused on the Buffered AFL [51], which has been shown to be more efficient than other AFL methods.

4.1.6 Buffered Asynchronous Federated Learning

Nguyen et al. [51] present a framework for Async-FL called FedBuff. Fedbuff uses a server-side buffer for aggregation incorporating client-level DP. More precisely, its implementation uses the Differentially Private Follow-the-Regularized-Leader (DP-FTRL) technique proposed in Kairouz et al. [52]. Unlike other AFL techniques, the server model is not immediately updated upon receiving each client's update; rather, the updates are stored in a buffer. A server update is only executed when the buffer includes K client updates. Nguyen et al. argue that K = 10 is the optimal buffer size; thus, we use the same value in the rest of this work.

4.2 HIERARCHICAL FEDERATED LEARNING MEETS DIFFERENTIAL PRIVACY

FL requires $K \leq N$ online federated clients per round to jointly learn a model with the central aggregator. This is a total of 2 actors, at 2 conceptual levels (client vs. server). In our setup, we assume the existence of 3 main actors, at 3 different (conceptual) levels of the hierarchy (as shown in Figure 18). At level 0 (i.e., the lowest level), we have the federated clients, who hold private data as in the status quo. At level 2 (i.e., the highest level), we have the central aggregator, again as in the status quo. The key difference lies in the middle; at level 1 (i.e., the intermediate level), we introduce a new entity called the super-node. The super-node is responsible for processing requests from a subset of the online clients in a particular region, henceforth termed a zone. The presence of level 1 introduces a hierarchical approach to performing FL (like those discussed earlier). The key distinction in our proposal is in how the super-nodes are chosen: they are elected by a pool of their peers, and thus are within the same trust boundary/region as their peers. Note that there can be many intermediary layers in practice, but we limit to one for simplicity of exposition.

© 2022-2025 FLUIDOS

Funded by Horizon Europe Framework Programme of the European Union

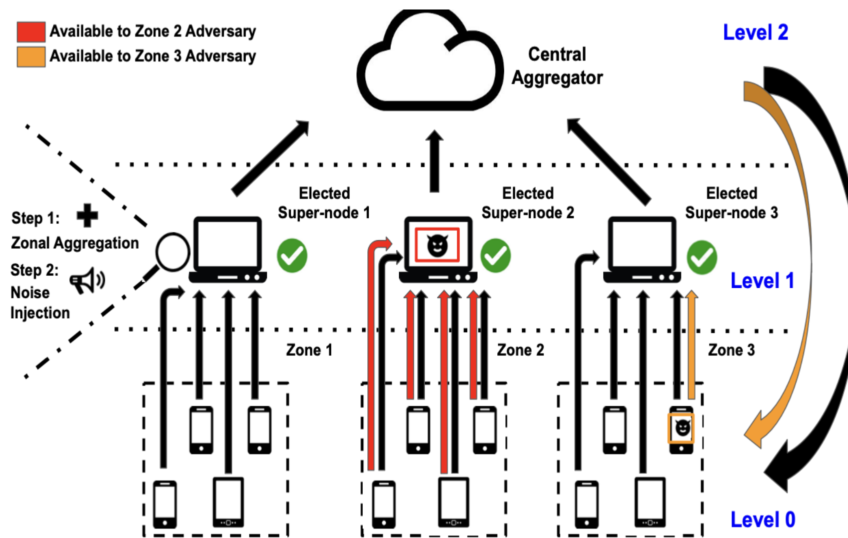


Figure 18 Architectural overview of our attack setup. Locally learnt gradients are perturbed with noise to provide DP guarantees. This ensures the gradients viewed by the aggregator are less noise. Adversarial nodes can be present at level 0 or 1

4.2.1 Impact on Scalability

Hierarchical FL improves the scalability of FL through the tiered architecture. The central aggregator is no longer required to process and respond to the federated clients directly (which are in the order of millions), but only interact with the super-nodes (which are 1-2 orders of magnitude lower). Additionally, the super-nodes and the clients in a particular zone can perform federated learning independently; this has the potential to further reduce the frequency of interaction with the central aggregator.

4.2.2 Impact on Privacy

In the status quo, formal privacy guarantees in the FL setting are provided through differentially private learning [53], where noise is added to the gradients during training [54]. This in turn is obtained through two mechanisms: noise addition at the central aggregator (i.e., CDP), or noise addition locally at each federated client (i.e., LDP). Empirical evidence suggests that the CDP mechanism will result in a more accurate (final) model learnt. However, it makes a crucial assumption: the central aggregator (where the noise is added) is considered trustworthy. The LDP mechanism removes this trust assumption at the expense of greater noise addition (at each client). To provide the best of both worlds, we propose hierarchical DP (or HDP). Based on the construction described above, clients aggregate their (zonal-level) updates through the super-nodes where calibrated noise is added to provide the DP guarantee. Through the remainder of this text, we will describe modifications made to the traditional approach to FL to achieve HDP.

Funded by Horizon Europe Framework Programme of the European Union

1: *Parameters*
2: user selection probability $q \in (0,1]$
3: per-user example cap $\hat{w} \in \mathbb{R}^+$
4: noise scale $z \in \mathbb{R}^+$
5: UserUpdate (for FedAvg or FedSGD)
6: ClipFn (FlatClip or PerLayerClip)
7: *Procedure:*
8: Initialize model θ^0
9: $w_k = \min(\frac{n_k}{\hat{w}}, 1)$ for all users k
10: **for** each round $t = 0, 1, 2, \ldots$ **do**
11: **for** each zone $i = 1, 2, \ldots, S$ **do**
12: $C_i^t \leftarrow$ (sample users with probability q)
13: $W = \sum_{k \in C_i^t} w_k$
14: **for** each user $k \in C_i^t$ **in parallel do**
15: $\Delta_k^{t+1} \leftarrow \text{UserUpdate}(k, \theta^t, \text{ClipFn})$
16: **end for**
17: $\Delta(i)^{t+1} = \begin{cases} \frac{\sum_{k \in C_i^t} w_k \Delta_k^{t+1}}{qW} & \text{for FlatClip} \\ \frac{\sum_{k \in C_i^t} w_k \Delta_k^{t+1}}{\max(qW_{min}, \sum_{k \in C_i^t} w_k)} & \text{for PerLayerClip} \end{cases}$
18: $\mathcal{S}_i \leftarrow$ (bound on $||\Delta_k||_2$ for ClipFn)
19: $\sigma_i \leftarrow \left\{ \frac{z\mathcal{S}_i}{qW} \text{ for FlatClip or } \frac{2z\mathcal{S}_i}{qW_{min}} \text{ for PerLayerClip} \right\}$
20: **end for**
21: $\theta^{t+1} \leftarrow \theta^t + \frac{1}{S} \sum_{i \in [S]} \left(\Delta(i)^{t+1} + \mathcal{N}(0, I\sigma_i^2) \right)$
22: **end for**

Figure 19 Modified FL mechanism with zonal privacy.

4.2.3 Proposed Algorithm

The algorithm presented in Figure 19 is heavily based on the approach taken by McMahan et al. [55], with some key modifications to incorporate hierarchical FL for S zones. From line 11, observe that gradient aggregation first occurs at a zonal level. This in turn leads to recalibration of various parameters (lines 12-19) required to provide the DP guarantee. Finally, the noised zonal gradients are averaged and aggregated in line 21. Note that in scenarios where uniform weighting is applied (i.e., the contribution of each client is the same), $w_k = 1$. Additionally, the variance of the noise to be added ($\boldsymbol{\sigma}_i$ in line 19) is calculated as a function of the online clients per zone (as noted by the denominator term: qW or qW_{min}), and not the total fraction of clients across all zones, as W is re-calibrated based on the selected clients per round per zone. The proposed approach calculates the sensitivity based on the clipping bound Si that is calculated across all clients across all zones. Obtaining this information in practice will require additional communication between the super-nodes. Thus, this is approximated through the existence of a global clipping bound C such that $C \geq \max_{i \in [S]} S_i$

4.2.3.1 Algorithmic Correctness

Funded by Horizon Europe Framework Programme of the European Union

Despite the presence of the intermediary level, algorithmic correctness must be preserved i.e., the aggregator must ensure that the final value that it obtains (and propagates) is the same as that obtained in the baseline FedAvg scenario [55]. To this end, we generalize the construction described earlier as follows: there are a total of S super-nodes in the intermediary level, each responsible for its own zone. Each zone has an equal number of clients. While this simplifying assumption enables easier calculation, this can be relaxed in practice. The federated clients within each zone clip their gradients (as is common in ML) and set the clipping norm to a global constant C, i.e., all participants (at any level of the hierarchy) are aware of the clipping threshold. Once these gradients are received, the super-node adds noise after performing the operation listed next. To achieve correctness, two operations can be performed at the intermediary level: (a) strict summation where the super-node aggregates the received gradients, or (b) averaging where the super-node aggregates and averages (by the number of clients in the zone) the gradients received. We describe both below.

Case 1: Only summation at the intermediary level. In this scenario, each of the S super-nodes is responsible for aggregating the values from K/S = m clients. Let ci denote the gradient update of client i. After clipping the gradients (using flat clipping), the value of each gradient is ci, where $\boldsymbol{\lambda}$ = max(1, ||ci||2 /C) and ||.||2 denotes the 2-norm. Thus, when the super-nodes perform summation, the maximum value obtainable is m.ci / $\boldsymbol{\lambda} \leq$ m.$\boldsymbol{\gamma}$ / $\boldsymbol{\lambda}$, where $-\boldsymbol{\gamma} <$ ci $\leq$ $\boldsymbol{\gamma}$,∀i.

Gaussian noise (required to provide DP) is added proportional to the l_2-sensitivity of the summation function, which in this case is proportional to 1/$\boldsymbol{\lambda}$. To ensure correctness in this case, the central aggregator is responsible for averaging (that is not performed by the super-nodes) and does so over the total of K clients.

Case 2: Averaging at the intermediary level. In this scenario, each of the S super-nodes is responsible for averaging the aggregated values. Thus, the average is calculated as: $\frac{1}{m} \cdot \sum_i \frac{c_i}{\lambda}$. The maximum achievable average is $\boldsymbol{\gamma}$/m·$\boldsymbol{\lambda}$, and thus the l2 sensitivity is proportional to 1/m·$\boldsymbol{\lambda}$. In such a situation, the central aggregator will have to average by the number of super-nodes (i.e., S) to preserve algorithmic correctness (since each super-node only averages by the number of clients it is responsible for).

Note that m > 1. Thus, the l_2-sensitivity in case 2 is smaller than in case 1. Thus, noise addition in case 2 would be lower than that of case 1, ergo utility obtained in case 1 would be lower than in case 2. Thus, in our hierarchical FL setup, we advocate designing hierarchies such that the super-nodes always perform averaging.

4.2.4 Evaluation

Funded by Horizon Europe
Framework Programme of the European Union

We implement our proposed approach using tensorflow (TF). We use a combination of tensorflow-federated v0.16.1 to provide the components required for FL, and tensorflow-privacy to provide the machinery required for private learning[12]. Such a framework allows us to calibrate for the clipping norm C and the noise multiplier z[13]. Any modifications to either of them will result in implications to both accuracy and privacy. To ensure correct accounting, we modify the accounting libraries in tensorflow-privacy to accurately reflect sampling probability and number of iterations (in this case, rounds). To evaluate the efficacy of our approach, we consider the datasets listed in Table 1. Only the EMNIST dataset is modified to exhibit the non-i.i.d properties that are commonly associated with FL. The datasets follow a standard 80:20 split (80% is used for training, and the remaining is used for validation). All our experiments were executed on a server with 2 NVidia Titan XP with 128 GB RAM and 48 CPU cores running Ubuntu 20.04.2. Due to computational constraints, and issues in tensorflow-federated related to GPU execution14, our experimental setup is conservative; we only perform a single run of each configuration and are unable to report error bars. However, we replicate the ecosystem chosen in prior work.

Dataset	Size	# Entries	# Classes	Total Clients
EMNIST [36]	$28 \times 28 \times 1$	382705	10	3383
CIFAR-10 [37]	$32 \times 32 \times 3$	60000	10	500
CIFAR-100 [38]	$32 \times 32 \times 3$	60000	100	500

Through our evaluation, we wish to answer the following question: Does the proposed hierarchical scheme truly provide advantageous privacy vs. utility trade-offs in comparison to using LDP and/or CDP? Recall that LDP provides more privacy, while CDP is known to provide a more utilitarian model. To this end, we evaluate our scheme based on the datasets and models specified in our proposed framework earlier. To ensure a fair comparison, we only report the privacy budget **ε** calculated at the central aggregator level in our results.

We first train without DP to get an estimate of the 2−norm of the gradients. This helps us estimate the clipping norm (C) to be used during DP training. We also observe the training duration (i.e., exact epoch number) at which the validation accuracy saturates. Once this is obtained, we configure the noise multiplier (z) to enable DP training and log the privacy

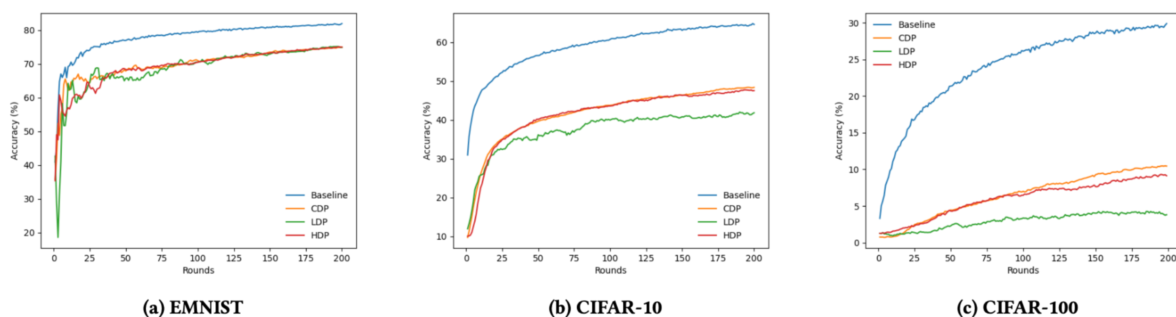
[12] https://github.com/tensorflow/federated/tree/v0.16.1 and https://github.com/tensorflow/privacy

[13] https://github.com/tensorflow/federated/issues/832

Funded by Horizon Europe Framework Programme of the European Union

expenditure (**ε**) at the end of training. Note that our training results, configurations, datasets, and models are similar to those of prior work in this area.

Privacy vs. Utility: Observe that different strategies of applying DP noise (i.e., central vs. local vs. hierarchical) result in differing values of privacy expenditure. To ensure a fair comparison of privacy vs. utility, we convert the privacy expenditure in all settings to that of the central DP privacy expenditure (using the formulation presented earlier). We can then plot the validation accuracy as a function of training duration, for all the datasets we consider. We also consider a scenario without any form of DP training enabled; this is our baseline. Note that for the hierarchical DP setting, we consider a scenario where there are S = 10 super-nodes. The results are plotted in Figure 20.



(a) EMNIST **(b) CIFAR-10** **(c) CIFAR-100**

Figure 20 We plot the validation accuracy of training with DP for 3 scenarios: (i) central DP, (ii) local DP, and (iii) the proposed hierarchical DP.

Observe that for our chosen configuration, learning simple tasks such as EMNIST (refer Figure 22 We plot the validation accuracy as a function of privacy for 3 scenarios using HDP: (i) K = 100, (ii) K=200, and (iii) K = 300.) with DP is achievable in all three settings, i.e., the accuracy degradation induced by LDP in comparison to CDP is minimal. However, as expected, HDP provides advantageous trade-offs in terms of accuracy. The results are more interesting for more complex datasets such as CIFAR-10 and CIFAR-100. First, observe that for the configuration we choose (i.e., N = 500, K = 100), baseline accuracy is ~ 63% and ~ 28% for the two CIFAR versions, respectively. Note that these values are comparable to those achieved by McMahan et al.12. Training with privacy (CDP) degrades this accuracy further. However, as expected, HDP can provide advantageous trade-offs in terms of privacy and accuracy. Note that CIFAR-100 is naively at least 10× more complex a learning task than CIFAR-10, and yet HDP achieves similar utility to CDP, and much higher than LDP.

Finally, methods. Contains corresponding values of the privacy budget achieved at the end of training. Observe that the privacy budget calculated at the end of HDP training is in-between that of the CDP and LDP case. In fact, with nearly a 67% decrease in privacy expenditure, HDP can achieve nearly the same accuracy as the CDP case across all 3 datasets we consider.

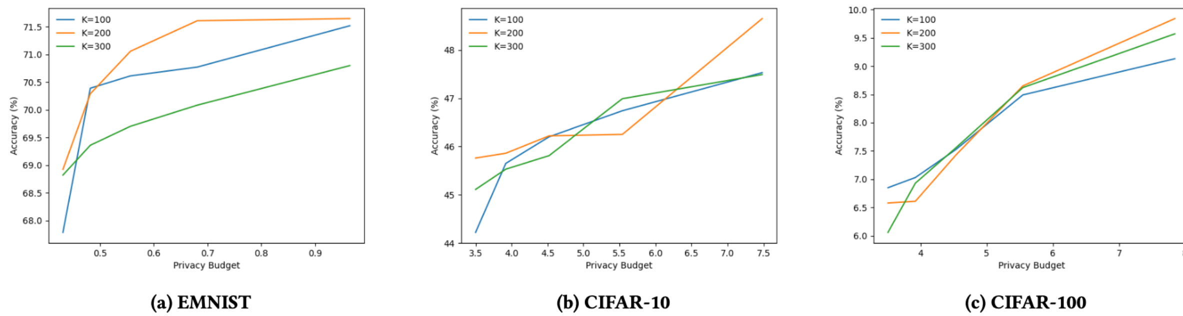
© 2022-2025 FLUIDOS

Funded by Horizon Europe Framework Programme of the European Union

Dataset	LDP	HDP	CDP
EMNIST	0.30	0.96	3.06
CIFAR-10	2.48	7.48	24.80
CIFAR-100	2.48	7.48	24.80

Figure 21 Privacy expenditure across datasets and DP methods.

The Influence of S: From our analysis, we can see that increasing the value of S makes the scheme more private (the privacy calculation is inversely proportional to the value of S). To better understand this hyperparameter, we consider an experimental setting where we vary the value of K to 100, 200, and 300. Across these 3 settings, we vary the value of S to one of {10, 20, 30, 40, 50} across all datasets. All other hyper-parameters were kept the same as in the earlier experiment. We then measure the validation accuracy and vary the configurations of C and z to obtain the privacy expenditure. We plot the relationship between the fully trained model's validation accuracy and the privacy budget expended to achieve it in Figure 22. Across all datasets we observe a common trend: the validation accuracy calculated increases as the privacy expenditure does. For the datasets we consider, increasing the value of K does not increase the validation accuracy substantially. This is not indicative of a more general trend; one would assume that increasing the number of parties (K) would result in a more accurate model.



(a) EMNIST (b) CIFAR-10 (c) CIFAR-100

Figure 22 We plot the validation accuracy as a function of privacy for 3 scenarios using HDP: (i) K = 100, (ii) K=200, and (iii) K = 300.

4.3 ASYNCHRONOUS FEDERATED LEARNING WITH LOCAL DIFFERENTIAL PRIVACY

Prior work in AFL has only studied CDP (as shown by Nguyen et al.). As mentioned earlier, this requires the server to be trusted with access to the clients' individual updates, and the

© 2022-2025 FLUIDOS

Funded by Horizon Europe Framework Programme of the European Union

responsibility of correctly adding DP noise to the aggregated updates. The first issue could, in theory, be mitigated using secure multi-party computation (i.e., performing secure aggregation so that the server can only recover the aggregates). However, this often comes with prohibitively high computational overhead. The second issue remains unmitigated. This prompts the need to build AFL techniques that satisfy Local DP (LDP), allowing clients to enjoy sound privacy guarantees vis-à-vis both the server and other adversarial clients.

In our approach, we introduce the first technique (to our knowledge) to guarantee LDP in AFL. We base our method on FedBuff (from Nguyen et al.) but modify it to achieve LDP guarantees without compromising utility.

Notes: We assume the presence of C distributed clients. Dc denotes client c's data, and nc is the number of data points on c, i.e., n = lDcl. **θ**c is the local model parameter. The optimization function of each client is:

$$f_c(\theta_c) = \frac{1}{n_c} \sum_{i \in D_c} l_i(x_i, y_i, \theta_c)$$

where l represents the loss function of the corresponding data point. Consequently, the aggregated optimization function is:

$$F(\theta) = \sum_{c=1}^{C} \frac{n_c}{C} f_c(\theta)$$

where **θ** is the model at the server side, and the goal is to find the optimized one (**θ**∗) for:

$$\theta_* = argminF(w)$$

4.3.1 Proposed Algorithm

In Figure 23 we detail our approach for achieving LDP in Buffered AFL. The FL process starts with the server generating a model with randomized weights. This model is then distributed to all participating clients. Each client independently trains this initial model on its respective dataset. During the training process, and to satisfy LDP, gradients are clipped to limit their magnitude, and each client adds Gaussian noise. The resulting privacy guarantees have been investigated in prior work and formally proven by Kim et al. [56] and Mahawaga Arachchige et al. [57].

Funded by Horizon Europe Framework Programme of the European Union

In AFL, there is no concept of rounds. Instead, each client sends its updates to the server as soon as they are available. The server aggregates these updates by inserting them into a buffer. When the buffer becomes full, the server performs a staleness control on the updates before aggregating them to ensure that all updates are consistent with the current model. This phase follows the same approach as Nguyen et al. and Huba et al. in handling stale updates. Once the check is completed, the server aggregates the updates to obtain a new model (with a new version denoted as r), which is then distributed to all clients for further training.

1. **Function** Main():
2. Initialize: model θ_0 Buffer B; **while** *the model is not converged* **do**
3. $c \leftarrow$ sample available clients
4. run CLIENT-TRAIN(*params*) asynchronously on c
5. **if** *receive client update* **then**
6. $\Delta_i \leftarrow$ update from client i
7. $B.insert(\Delta_i)$
8. **end if**
9. **if** $B.isFull()$ **then**
10. apply_staleness_control(B)
11. $\Delta^{-r} \leftarrow$ average_client_updates(B)
12. $\theta^{r+1} \leftarrow \theta^r - \eta_s \Delta^{-r}$
13. $B.empty(), \Delta^{-r} \leftarrow 0, r \leftarrow r+1$
14. **end if**
15. **end while**
16. **return** θ
17. **Function** CLIENT-TRAIN(*clipping norm S, dataset D, sampling probability p, noise magnitude σ, client learning rate η_c, Iterations E, loss function $L(\theta(x), y)$*):
18. Initialize θ_0
19. **for** *each local epoch i from 1 to E* **do**
20. **for** $(x, y) \in$ *random batch from dataset D with probability p* **do**
21. $g_i = \nabla_\theta L(\theta_i; (x, y))$
22. **end for**
23. $g_{batch} = \frac{1}{p \cdot D}(\Sigma_{i \in batch} g_i \cdot min(1, \frac{S}{\|g_i\|_2}) + N(0, \sigma^2 I))$
24. $\theta_{i+1} = \theta_i - \eta_c(g_{batch})$
25. **end for**
26. **return** θ_c

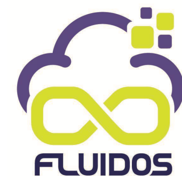
Figure 23 Local Differential Privacy in Buffered AFL (FedBuff+LDP)

4.3.2 Evaluation

Our experiments use three different datasets: CIFAR-10 [58], CINIC-10 [59] and reddit-comments[15]. CIFAR-10 includes 60,000 labelled images (50,000 training and 10,000 testing), each depicting one of ten object classes, with 6,000 images per class. CINIC-10 has 10 classes

[15] https://bit.ly/google-reddit-comms

Funded by Horizon Europe Framework Programme of the European Union



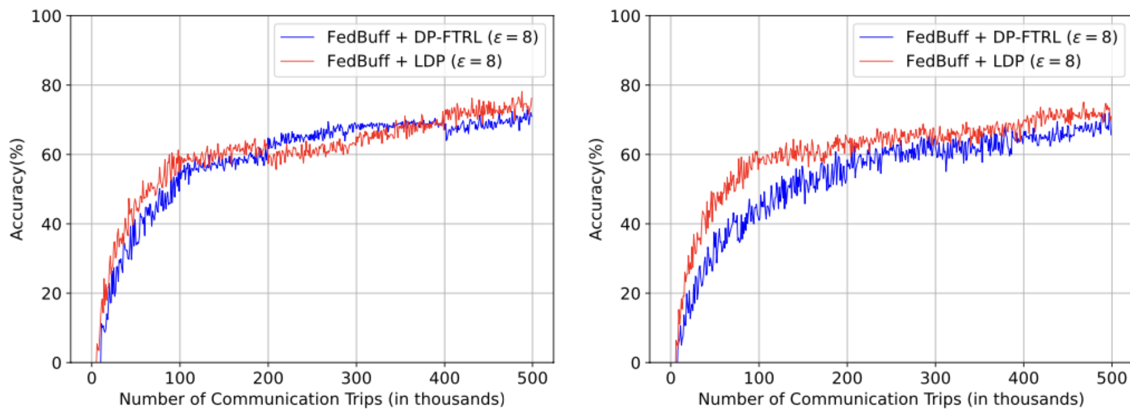
and 270,000 images, 180,000 of which are used for training and 90,000 for testing. To train on CIFAR-10 and CINIC-10 datasets, we use the lightweight ResNet18 CNN model [60]. We also consider a word prediction task using the Reddit-comments dataset. We use a model with a two-layer Long Short-Term Memory (LSTM) and 10 million parameters trained on a chosen month (September 2019) from the Reddit-comments dataset, and we filter users with several posts between 350 and 500. Our training setup is like past work [61], and our dictionary is restricted to the 30K most frequent words (instead of 50K) to speed up training and boost model accuracy.

To partition the CIFAR-10 and CINIC-10 datasets, we follow the approach of past work of Nguyen et al. and use a Dirichlet distribution with parameter 0.1 to divide the dataset among 5000 non-iid clients. For Reddit-comments, the filtered-out users are recognized as clients with their posts as their training data. Following Nguyen et al., we set the buffer size to 10. We experiment with different values of concurrency, i.e., the number of clients training at a particular time. We sample the staleness of clients from a half-normal distribution with $\boldsymbol{\sigma}$ = 0.5.

In Figure 24, we report the model performance with respect to test accuracy of FedBuff+LDP (our algorithm) compared to FedBuff+DP-FTRL (providing Central DP) across an increasing number of communication trips, which encompass download, computation, and upload.

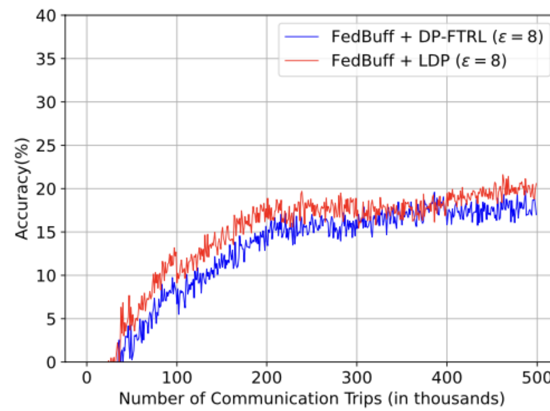
© 2022-2025 FLUIDOS

Funded by Horizon Europe
Framework Programme of the European Union



(a) CIFAR-10

(b) CINIC-10



(c) Reddit-comments

Figure 24 Main task accuracy in buffered async FL with LDP in different communication trips.

Then, in Figure 25, we present the accuracy results of each model for different privacy budgets, for the three datasets and models and the two tested methods.

Funded by Horizon Europe Framework Programme of the European Union

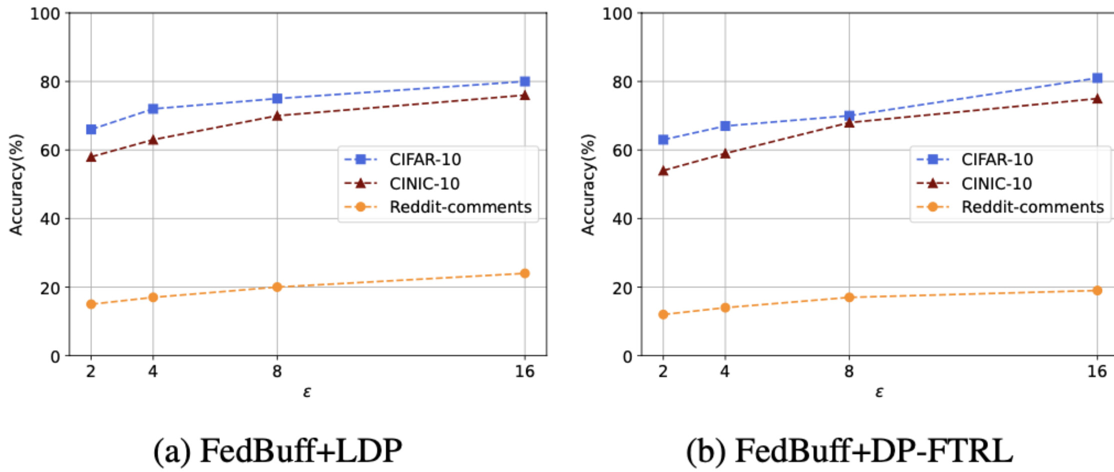


Figure 25 Main task accuracy in buffered async FL with LDP for three datasets with varying privacy budgets **ε** (lower **ε** provides better privacy).

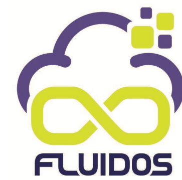
Overall, our experiments show that, under the same privacy budget (i.e., with the same epsilon values), FedBuff+LDP provides better accuracy than FedBuff DP-FTRL. It also does this faster, i.e., FedBuff+LDP enables the construction of a global model that achieves higher accuracy faster with respect to communication trips. Note that using the same epsilon values in Local vs. Central DP variants does not necessarily imply the same level of privacy since these values have different meanings in either setting.

Finally, Figure 26 shows the number of communication trips needed to reach a certain accuracy for the three datasets. The results are averaged over five random runs, and the standard deviation is computed based on these runs. Again, we observe that FedBuff+LDP picks up better accuracy and faster – i.e., in fewer trips – than FedBuff+DP-FTRL.

Dataset	Accuracy	FedBuff w/LDP	FedBuff w/DP-FTRL
CIFAR-10	60%	137.8 ± 3.9	195.2 ± 4.2
CINIC-10	60%	154.1 ± 6.2	233.7 ± 5.3
Reddit-comments	15%	176.1 ± 9.1	214.8 ± 7.5

Figure 26 Average ± standard deviation number of communication trips (in thousands) to reach a target accuracy in the three datasets with **ε** = 8 and K = 10. Standard deviation is computed over 5 random runs of each setting with different seeds.

Funded by Horizon Europe Framework Programme of the European Union



REFERENCES

[1] Santos, José, Chen Wang, Tim Wauters and Filip De Turck. "Diktyo: Network-Aware Scheduling in Container-based Clouds." IEEE Transactions on Network and Service Management (2023).

[2] Guim, Francesc, Thijs Metsch, Hassnaa Moustafa, Timothy Verrall, David Carrera, Nicola Cadenelli, Jiang Chen, David Doria, Chadie Ghadie and Raül González Prats. "Autonomous Lifecycle Management for Resource-Efficient Workload Orchestration for Green Edge Computing." IEEE Transactions on Green Communifcations and Networking 6 (2022): 571-582.

[3] Song, Shudian, Shuyue Ma, Jingmei Zhao, Feng Yang and Linbo Zhai. "Cost-efficient multi-service task offloading scheduling for mobile edge computing." Applied Intelligence 52 (2021): 4028-4040.

[4] Tamiru, Mulugeta Ayalew, Guillaume Pierre, Johan Tordsson and Erik Elmroth. "mck8s: An orchestration platform for geo-distributed multi-cluster environments." 2021 International Conference on Computer Communications and Networks (ICCCN) (2021): 1-10.

[5] Castellano, Gabriele, Flavio Esposito and Fulvio Risso."A Distributed Orchestration Algorithm for Edge Computing Resources with Guarantees." IEEE INFOCOM 2019 - IEEE Conference on Computer Communications (2019): 2548-2556.

[6] Intel: Intent Driven Orchestration, 2022: https://github.com/intel/intent-driven-orchestration [Accessed: 24/10/2023]

[7] Morichetta, Andrea, Nikolaus Spring, Philipp Raith and Schahram Dustdar. "Intent-based Management for the Distributed Computing Continuum." 2023 IEEE International Conference on Service-Oriented System Engineering (SOSE) (2023): 239-249.

[8] Wu, Chaofeng, Shingo Horiuchi, Kenji Murase, Hiroaki Kikushima and Kenichi Tayama. "An Intent-driven DaaS Management Framework to Enhance User Quality of Experience." ACM Transactions on Internet Technology 22 (2022): 1-25.

[9] Metsch, Thijs, Magdalena Viktorsson, Adrian Hoban, Monica Vitali, Ravi Iyer and Erik Elmroth. "Intent-Driven Orchestration: Enforcing Service Level Objectives for Cloud Native Deployments." SN Computer Science 4 (2023): 1-13.

[10] Zhong, Zhiheng, Minxian Xu, Maria Alejandra Rodriguez, Chengzhong Xu and Rajkumar Buyya. "Machine Learning-based Orchestration of Containers: A Taxonomy and Future Directions." ACM Computing Surveys (CSUR) 54 (2021): 1-35.

[11] Han, Yiwen, Shihao Shen, Xiaofei Wang, Shiqiang Wang and Victor C. M. Leung. "Tailored Learning-Based Scheduling for Kubernetes-Oriented Edge-Cloud System." IEEE INFOCOM 2021 - IEEE Conference on Computer Communications (2021): 1-10.

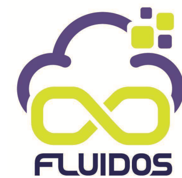
[12] Lou, Jiong, Zhiqing Tang and Weijia Jia. "Energy-Efficient Joint Task Assignment and Migration in Data Centers: A Deep Reinforcement Learning Approach." IEEE Transactions on Network and Service Management 20 (2023): 961-973.

[13] Iftikhar, Sundas, Mirza Mohammad Mufleh Ahmad, Shreshth Tuli, Deepraj Chowdhury, Minxian Xu, Sukhpal Singh Gill and Steve Uhlig. "HunterPlus: AI based energy-efficient task scheduling for cloud-fog computing environments." Internet Things 21 (2022): 100667.

[14] Li, Qian, Bin Li, Pietro Mercati, Ramesh Illikkal, Charlie Tai, Michael Kishinevsky and Christos Kozyrakis. "RAMBO: Resource Allocation for Microservices Using Bayesian Optimization." IEEE Computer Architecture Letters 20 (2021): 46-49.

[15] Samanta, Amit, Yong Li, and Flavio Esposito. "Battle of Microservices: Towards Latency-Optimal Heuristic Scheduling for Edge Computing." 2019 IEEE Conference on Network Softwarization (NetSoft) (2019): 223-227.

Funded by Horizon Europe Framework Programme of the European Union



[16] Pallewatta, Samodha, Vassilis Kostakos and Rajkumar Buyya. “Microservices-based IoT Application Placement within Heterogeneous and Resource Constrained Fog Computing Environments.” Proceedings of the 12th IEEE/ACM International Conference on Utility and Cloud Computing (2019): pp. 71-81.

[17] Aryal, Ram Govinda and Jörn Altmann. “Dynamic application deployment in federations of clouds and edge resources using a multiobjective optimization AI algorithm.” 2018 Third International Conference on Fog and Mobile Edge Computing (FMEC) (2018): 147-154.

[18] Senjab, K., Abbas, S., Ahmed, N. et al. “A survey of Kubernetes scheduling algorithms,” J Cloud Comp, 12, 87 (2023), pp. 1-26.

[19] Rejiba Z., and Chamanara J., "Custom scheduling in Kubernetes: a survey on common problems and solution approaches," ACM Comput. Surv., 55-7, 2023, Article 151, pp. 1-37.

[20] Carrión C., “Kubernetes scheduling: taxonomy, ongoing issues and challenges,” ACM Comput Surv, 55(7), 2022, Article 138, pp. 1–37

[21] Scheduling SIG, https://github.com/kubernetes/community/tree/master/sig-scheduling

[22] Ogbuachi MC, Gore C, Reale A, et al., “Context-aware K8S scheduler for real time distributed 5G edge computing applications,”, in Proc. Int. Conf. Softw., Telecom and Comp. Netw. (SoftCOM), 2019, pp. 1-6.

[23] Haja D, Szalay M, Sonkoly B, et al., “Sharpening Kubernetes for the Edge,” in Proc. ACM SIGCOMM Conf., 2019, pp. 136–137.

[24] Wojciechowski L K., Opasiak K., and Latusek, J. et al., “NetMARKS: Network metrics-aware Kubernetes scheduler powered by service mesh,”, in Proc. IEEE Conf. Comp. Comm., 2021, pp. 1-9.

[25] Oleghe O., “Container placement and migration in edge computing: concept and scheduling models,” IEEE Access, 9, 2021, pp. 68028–68043.

[26] Li D, Wei Y, Zeng B, “A dynamic I/O sensing scheduling scheme in Kubernetes,” ACM Int. Conf., 2020, pp 14–19.

[27] Covington, Paul, Jay K. Adams and Emre Sargin. “Deep Neural Networks for YouTube Recommendations.” Proceedings of the 10th ACM Conference on Recommender Systems (2016)

[28] Shi, W., Cao, J., Zhang, Q., Li, Y., & Xu, L. (2016). Edge computing: Vision and challenges. IEEE internet of things journal, 3(5), 637-646.

[29] Garcia Lopez, P., Montresor, A., Epema, D., Datta, A., Higashino, T., Iamnitchi, A., ... & Riviere, E. (2015). Edge-centric computing: Vision and challenges. ACM SIGCOMM Computer Communication Review, 45(5), 37-42.

[30] Baresi, L., Mendonça, D. F., Garriga, M., Guinea, S., & Quattrocchi, G. (2019). A unified model for the mobile-edge-cloud continuum. ACM Transactions on Internet Technology (TOIT), 19(2), 1-21.

[31] Tomarchio, O., Calcaterra, D., & Modica, G. D. (2020). Cloud resource orchestration in the multi-cloud landscape: a systematic review of existing frameworks. Journal of Cloud Computing, 9, 1-24.

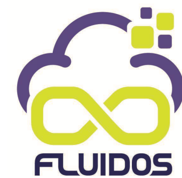
[32] https://github.com/intel/platform-aware-scheduling/tree/master

[33] Marchese, A., & Tomarchio, O. (2022, May). Network-aware container placement in cloud-edge kubernetes clusters. In 2022 22nd IEEE International Symposium on Cluster, Cloud and Internet Computing (CCGrid) (pp. 859-865). IEEE.

[34] Carmona-Cejudo, E., Iadanza, F., & Siddiqui, M. S. (2022, April). Optimal offloading of Kubernetes pods in three-tier networks. In 2022 IEEE Wireless Communications and Networking Conference (WCNC) (pp. 280-285). IEEE.

[35] Cynthia Dwork, Aaron Roth, et al. The algorithmic foundations of differential privacy. Foundations

Funded by Horizon Europe
Framework Programme of the European Union



and Trends in Theoretical Computer Science, 9(3-4):211–407, 2014.

[36] Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Aguera y Arcas. Communication-efficient learning of deep networks from decentralized data. In Artificial Intelligence and Statistics, pages 1273–1282. PMLR, 2017.

[37] H. Brendan McMahan, Daniel Ramage, Kunal Talwar, and Li Zhang. Learning differentially private language models without losing accuracy. arXiv preprint arXiv:1710.06963, 2017.

[38] Robin C Geyer, Tassilo Klein, and Moin Nabi. Differentially private federated learning: A client level perspective. arXiv preprint arXiv:1712.07557, 2017.

[39] Keith Bonawitz, Vladimir Ivanov, Ben Kreuter, Antonio Marcedone, H Brendan McMahan, Sarvar Patel, Daniel Ramage, Aaron Segal, and Karn Seth. Practical secure aggregation for federated learning on user-held data. arXiv preprint arXiv:1611.04482, 2016.

[40] Keith Bonawitz, Vladimir Ivanov, Ben Kreuter, Antonio Marcedone, H Brendan McMahan, Sarvar Patel, Daniel Ramage, Aaron Segal, and Karn Seth. Practical secure aggregation for privacy-preserving machine learning. In Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, pages 1175–1191, 2017.

[41] Stacey Truex, Nathalie Baracaldo, Ali Anwar, Thomas Steinke, Heiko Ludwig, Rui Zhang, and Yi Zhou. A hybrid approach to privacy-preserving federated learning. In Proceedings of the 12th ACM Workshop on Artificial Intelligence and Security, pages 1–11, 2019

[42] Stacey Truex, Ling Liu, Ka-Ho Chow, Mehmet Emre Gursoy, and Wenqi Wei. Ldp-fed: federated learning with local differential privacy. In Proceedings of the Third ACM International Workshop on Edge Systems, Analytics and Networking, pages 61–66, 2020

[43] M Salehi Heydar Abad, Emre Ozfatura, Deniz Gunduz, and Ozgur Ercetin. Hierarchical federated learning across heterogeneous cellular networks. In ICASSP 2020-2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), pages 8866–8870. IEEE, 2020.

[44] Jinliang Yuan, Mengwei Xu, Xiao Ma, Ao Zhou, Xuanzhe Liu, and Shangguang Wang. Hierarchical federated learning through LAN-WAN orchestration. arXiv preprint arXiv:2010.11612, 2020.

[45] Christopher Briggs, Zhong Fan, and Peter Andras. Federated learning with hierarchical clustering of local updates to improve training on non-iid data. In 2020 International Joint Conference on Neural Networks (IJCNN), pages 1–9. IEEE, 2020.

[46] J. Wang, Z. Charles, Z. Xu, G. Joshi, H. B. McMahan, M. Al-Shedivat, G. Andrew, S. Avestimehr, K. Daly, D. Data, et al. A field guide to federated optimization. arXiv:2107.06917, 2021.

[47] Z. Chai, Y. Chen, A. Anwar, L. Zhao, Y. Cheng, and H. Rangwala. FedAT: a high-performance and communication-efficient federated learning system with asynchronous tiers. In SC, 2021.

[48] M. van Dijk, N. V. Nguyen, T. N. Nguyen, L. M. Nguyen, Q. Tran-Dinh, and P. H. Nguyen. Asynchronous federated learning with reduced number of rounds and with differential privacy from less aggregated gaussian noise. arXiv:2007.09208, 2020.

[49] W. Wu, L. He, W. Lin, R. Mao, C. Maple, and S. Jarvis. Safa: A semi- asynchronous protocol for fast federated learning with low overhead. IEEE Transactions on Computers, 70(5):655–668, 2020.

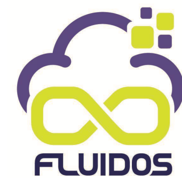
[50] C. Xie, S. Koyejo, and I. Gupta. Asynchronous federated optimization. In OPT, 2022.

[51] J. Nguyen, K. Malik, H. Zhan, A. Yousefpour, M. Rabbat, M. Malek, and D. Huba. Federated learning with buffered asynchronous aggregation. In AISTATS, 2022.

[52] P. Kairouz, B. McMahan, S. Song, O. Thakkar, A. Thakurta, and Z. Xu. Practical and private (deep) learning without sampling or shuffling. In ICML, 2021a.

[53] Martin Abadi,Andy Chu,Ian Goodfellow,H Brendan McMahan,Ilya Mironov, Kunal Talwar, and Li

Funded by Horizon Europe Framework Programme of the European Union



Zhang. Deep learning with differential privacy. In Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, pages 308–318, 2016.

[54] H Brendan McMahan, Daniel Ramage, Kunal Talwar, and Li Zhang. Learning differentially private language models without losing accuracy. arXiv preprint arXiv:1710.06963, 2017.

[55] Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Aguera y Arcas. Communication-efficient learning of deep networks from decentralized data. In Artificial Intelligence and Statistics, pages 1273–1282. PMLR, 2017.

[56] M. Kim, O. Günlü, and R. F. Schaefer. Federated learning with local differential privacy: Trade-offs between privacy, utility, and communication. In ICASSP 2021-2021 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), pages 2650–2654. IEEE, 2021.

[57] P. C. Mahawaga Arachchige, D. Liu, S. Camtepe, S. Nepal, M. Grobler, P. Bertok, and I. Khalil. Local differential privacy for federated learning. In Computer Security–ESORICS 2022: 27th European Symposium on Research in Computer Security, Copenhagen, Denmark, September 26–30, 2022, Proceedings, Part I, pages 195–216. Springer, 2022.

[58] A. Krizhevsky. Learning multiple layers of features from tiny images. Master's the- sis, University of Toronto, 2009. URL https://www.cs.toronto.edu/~kriz/ learning-features-2009-TR.pdf

[59] L. N. Darlow, E. J. Crowley, A. Antoniou, and A. J. Storkey. CINIC-10 is not ImageNet or CIFAR-10. arXiv:1810.03505, 2018.

[60] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In IEEE CVPR, pages 770–778, 2016.

[61] E. Bagdasaryan, A. Veit, Y. Hua, D. Estrin, and V. Shmatikov. How to backdoor federated learning. In AISTATS, 2020.

[62] C. Lauwers, C. Noshpitz, and C. Curescu, "Tosca simple profile in yaml version 1.3."

[63] P. Lipton, D. Palma, M. Rutkowski, and D. A. Tamburri, "Tosca solves big problems in the cloud and beyond!," /Ieee cloud computing/, p. 1, 2018, doi: [10.1109/MCC.2018.111121612].

[64] A. Nejc Bat and L. Korbar, "Xopera: Get your orchestra(tor) pitch perfect.

[65] D. Tamburri, W.-J. Heuvel, C. Lauwers, P. Lipton, D. Palma, and M. Rutkowski, "Tosca-based intent modelling: goal-modelling for infrastructure-as-code," Sics software-intensive cyber-physical systems, vol. 34, 2019, doi: [10.1007/s00450-019-00404-x].

[66] M. D. Mascarenhas and R. S. Cruz, "Int2it: An intent-based tosca it infrastructure management platform," in /2022 17th Iberian conference on information systems and technologies (cisti)/, 2022, pp. 1–7.

[67] A. S. Jacobs et al., "Hey, lumi! using natural language for Intent-Based network management," in /2021 usenix annual technical conference (usenix atc 21)/, Jul. 2021, pp. 625–639.

[68] A. S. Jacobs, R. J. Pfitscher, R. A. Ferreira, and L. Z. Granville, "Refining network intents for self-driving networks," in /Proceedings of the afternoon workshop on self-driving networks/, 2018, pp. 15–21.

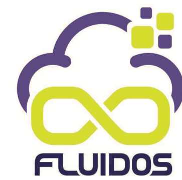
[69] R. Soulé et al., "Merlin: A language for managing network resources," Ieee/acm transactions on networking, vol. 26, no. 5, pp. 2188–2201, 2018.

[70] M. Riftadi and F. Kuipers, "P4i/o: Intent-based networking with P4," in /2019 ieee conference on network softwarization (netsoft)/, 2019, pp. 438–443.

[71] A. Angi, A. Sacco, F. Esposito, G. Marchetto, and A. Clemm, "Nlp4: An architecture for intent-driven data plane programmability," in 2022 ieee 8th international conference on network softwarization (netsoft), 2022, pp. 25–30.

[72] Valenza, Fulvio, and Antonio Lioy. "User-oriented Network Security Policy Specification." J. Internet

Funded by Horizon Europe
Framework Programme of the European Union



Serv. Inf. Secur. 8.2 (2018): 33-47.

[73] Zarca, Alejandro Molina, et al. "Virtual IoT HoneyNets to mitigate cyberattacks in SDN/NFV-enabled IoT networks." IEEE Journal on Selected Areas in Communications 38.6 (2020): 1262-1277

[74] Molina Zarca, Alejandro, et al. "Semantic-aware security orchestration in SDN/NFV-enabled IoT systems." Sensors 20.13 (2020): 3622

[75] J. M. B. Murcia, J. F. P. Zarca, A. M. Zarca and A. Skármeta, "By-default Security Orchestration on distributed Edge/Cloud Computing Framework," 2023 IEEE 9th International Conference on Network Softwarization (NetSoft), Madrid, Spain, 2023, pp. 504-509, doi: 10.1109/NetSoft57336.2023.10175478.

[76] Cao, Haotong, Han Hu, Zhicheng Qu, and Longxiang Yang. "Heuristic solutions of virtual network embedding: A survey." China Communications 15, no. 3 (2018): 186-219.

[77] Centofanti, C., Tiberti, W., Marotta, A., Graziosi, F., & Cassioli, D. (2023, June). Latency-Aware Kubernetes Scheduling for Microservices Orchestration at the Edge. In 2023 IEEE 9th International Conference on Network Softwarization (NetSoft) (pp. 426-431). IEEE.

Funded by Horizon Europe
Framework Programme of the European Union