ProvLight: Efficient Workflow Provenance Capture on the Edge-to-Cloud Continuum

Daniel Rosendo*, Marta Mattoso†, Alexandru Costan*, Renan Souza‡, Débora Pina†,
Patrick Valduriez§, Gabriel Antoniu*
*University of Rennes, Inria, CNRS, IRISA - Rennes, France
{daniel.rosendo, alexandru.costan, gabriel.antoniu}@inria.fr
†Federal University of Rio de Janeiro, Brazil
{marta, dbpina}@cos.ufrj.br
‡Oak Ridge National Laboratory, USA
souzar@ornl.gov
§University of Montpellier, Inria, CNRS, LIRMM - Montpellier, France
patrick.valduriez@inria.fr

***Abstract*—Modern scientific workflows require hybrid infrastructures combining numerous decentralized resources on the IoT/Edge interconnected to Cloud/HPC systems (aka the *Computing Continuum*) to enable their optimized execution. Understanding and optimizing the performance of such complex Edge-to-Cloud workflows is challenging. Capturing the provenance of key performance indicators, with their related data and processes, may assist in understanding and optimizing workflow executions. However, the capture overhead can be prohibitive, particularly in resource-constrained devices, such as the ones on the IoT/Edge.**

To address this challenge, based on a performance analysis of existing systems, we propose ProvLight, a tool to enable efficient provenance capture on the IoT/Edge. We leverage simplified data models, data compression and grouping, and lightweight transmission protocols to reduce overheads. We further integrate ProvLight into the E2Clab framework to enable workflow provenance capture across the Edge-to-Cloud Continuum. This integration makes E2Clab a promising platform for the performance optimization of applications through reproducible experiments.

We validate ProvLight at a large scale with synthetic workloads on 64 real-life IoT/Edge devices in the FIT IoT LAB testbed. Evaluations show that ProvLight outperforms state-of-the-art systems like ProvLake and DfAnalyzer in resource-constrained devices. ProvLight is 26—37x faster to capture and transmit provenance data; uses 5—7x less CPU; 2x less memory; transmits 2x less data; and consumes 2—2.5x less energy. ProvLight [1] and E2Clab [2] are available as open-source tools.

***Index Terms*—Provenance, Lineage, Workflows, Edge, IoT, Computing Continuum.**

I. Introduction

Data processing and Artificial Intelligence (AI) workflows can no longer rely on traditional approaches (due to resource usage, latency, and privacy constraints) [3] that send all data to centralized and distant Cloud datacenters for processing or AI model training [4]. Instead, they need to leverage hybrid (decentralized) approaches that take advantage of the numerous resources close to the data generation sites (*i.e.*, on the edge of the network) to promptly extract insights [5] and satisfy the ultra-low latency requirements of applications.

This hybrid approach contributes to the emergence of the *Computing Continuum* [6] (or the *Edge-to-Cloud Continuum* or the *Transcontinuum*). It seamlessly combines resources and services at the center of the network (*e.g.*, in Cloud datacenters), at its edge, and *in-transit*, along the data path. Typically, data is first generated and preprocessed (*e.g.*, model training with local data) on IoT/Edge devices. Then, data is transferred to (HPC-enabled) Clouds for Big Data analytics, AI model training, and global simulations. For instance, in Federated Learning (FL) model training, a central Cloud server collects data (model updates) from multiple decentralized Edge devices, then it generates a single accurate global inference model.

Due to the complexity incurred by application deployments on such highly distributed and heterogeneous Edge-to-Cloud infrastructures, realizing the Computing Continuum vision in practice is challenging. Deploying, analyzing, and reproducing performance trade-offs and optimizing large-scale, real-life applications on such infrastructures is difficult [3]. It requires configuring a myriad of system-specific parameters (*e.g.,* from AI and Big Data systems) and reconciling many requirements or constraints in terms of energy consumption, network efficiency, and hardware resource usage, to cite a few [7]. In recent works, these challenges have been mainly explored by systems like Pegasus [8], E2Clab [9], Delta [10].

The process of understanding, optimizing, and reproducing complex Edge-to-Cloud workflows may be assisted by **provenance data capture**. "Provenance data" refer to a record trail that accounts for **the origin of a piece of data** together with descriptions of the computational processes that assist in explaining **how and why it was generated** [11]. **Capturing provenance data during workflow execution** helps users in tracking inputs, outputs, and processing history, allowing them to steer workflows precisely [12].

For instance, considering a Federated Learning model training workflow executed on distributed devices on the Edge, the captured data during model training helps answer questions like: *(i) What are the elapsed time and the training loss in the latest epoch for each hyperparameter combination? [13], [14]* or *(ii) Retrieve the hyperparameters which obtained the*

3 best accuracy values for model m*? [14], [15].* Answering such queries helps to analyze hyperparameter values related to the training stages and to adjust them for better-quality results.

A. Challenges and Novelty

Overhead in provenance systems is a critical problem that must be assessed [16]. Many other contributions in provenance systems evaluate the overhead, such as [17], [18]. Overhead is even more critical in edge devices because of resource constraints and power consumption. For this reason, we decided to focus on evaluating overhead in our work. In [19], leading database researchers discussed the challenges of deploying services considering disaggregation and high heterogeneity of resources in hybrid cloud infrastructures. In [20], the authors describe challenges related to capturing provenance on the Edge-to-Cloud Continuum.

The main state-of-the-art provenance systems were designed to run on Cloud/HPC infrastructures. We highlight that we have not found in the literature reference systems tailored for IoT/Edge devices. Therefore, this work refers to systems well-known for their low provenance capture overhead in Cloud/HPC, such as DfAnalyzer [18], ProvLake [17], and PROV-IO [21]. We also include Komadu [22] in our analysis, as it is also compared within the aforementioned works.

Enabling provenance data capture with low overhead in resource-constrained IoT/Edge devices **cannot be easily achieved** by existing provenance systems, calling for practical solutions beyond the state-of-the-art. For instance, it requires the design and development of novel capture approaches focusing on the hardware limitations of IoT/Edge devices, as proposed in this work.

B. Contributions

We make the following contributions:

1) The first research question we aim to answer is: ***How Do the Existing Provenance Systems Perform in IoT/Edge Devices?*** We address this research question by providing an experimental evaluation of existing provenance systems along with a detailed discussion in Section III.
2) As our experiments concluded that the state-of-the-art systems present high overheads to capture provenance data in IoT/Edge devices, **we propose a novel workflow provenance data capture approach tailored for resource-limited IoT/Edge devices**, that addresses the limitations found in the state of the art (Section IV). **ProvLight is an open-source implementation** of this approach (available at [1]), following the W3C PROV-DM recommendations.
3) **An integration of ProvLight within the E2Clab** automatic deployment and performance optimization framework. This **enables provenance data capture across the Computing Continuum** for hybrid workflows deployed on both IoT/Edge and Cloud/HPC infrastructures. To the best of our knowledge, this enhanced version of E2Clab is **the first framework to support the end-to-end provenance data capture** of complex workflows executed on the Edge-to-Cloud Continuum (Section V). This integration with E2Clab is an open-source tool available at [2]. We highlight that, **ProvLight may be easily integrated into other deployment and performance optimization systems/frameworks**.
4) **A large-scale experimental validation** of ProvLight with synthetic workloads **on 64 real-life IoT devices** (from the FIT IoT LAB [23] testbed) and **Cloud resources** (from the Grid'5000 [24] testbed). Experimental evaluations show that ProvLight outperforms (*i.e.,* lower capture overhead) DfAnalyzer and ProvLake systems in terms of capture time, CPU and memory usage, network usage, and power consumption (Section VI).

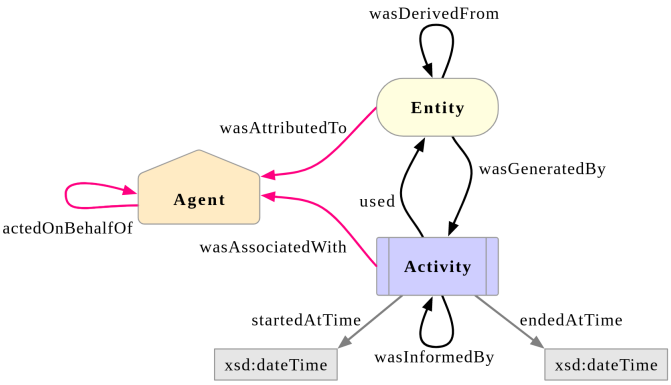


Fig. 1: PROV-DM: The W3C PROV Data Model [28].

II. Background

This work focuses on provenance systems **leveraging the user-defined capture approach** [25], [26]. This approach allows users to define what to capture by workflow script instrumentation through capture libraries. We highlight that script instrumentation (*e.g.*, adding logging calls) is a common practice in distributed systems, particularly to assist debugging. In provenance capture, many other approaches rely on script instrumentation [17], [26], [27]. A good practice to promote data interoperability is that such libraries should follow provenance specifications like the PROV-DM recommendation, as an example. Finally, the provenance capture of Edge-to-Cloud workflows is a new topic that requires automatic deployment tools like E2Clab.

A. PROV-DM: The PROV Data Model

PROV [29] is a specification to interchange provenance information. PROV-DM [28] is the data model for the W3C provenance family of specifications. It aims to promote data interoperability from provenance management systems. Provenance systems such as DfAnalyzer [18], ProvLake [17], PROV-IO [21], Komadu [22], among many others, follow the PROV-DM model.

Figure 1 illustrates the core elements of PROV-DM and their relationships. PROV-DM provides an abstract representation of provenance data derivations. Briefly described, the core elements are: *(i) Agent:* refers to tools invoked on behalf

of users (*e.g.,* software); *(ii) Activity:* refers to tasks (*e.g.,* processing steps); and *(i) Entity:* refers to data objects (*e.g.,* files, input parameters, etc.). Our capture approach also follows the PROV-DM recommendation.

B. Capturing Provenance for Edge-to-Cloud Workflows

1) ***Edge-to-Cloud Computing Continuum****:* Edge infrastructures refer to computing and storage resources located where the data originated. They consist of numerous smart devices sensing *"what"* is happening in the environment and generating potentially huge data streams at potentially high rates [3]. The Edge computing paradigm aims to push intelligence to those devices and extract value from data in real-time to improve response times while preserving privacy and security (critical data is analyzed locally).

Cloud infrastructures provide virtually "unlimited" computing and storage resources used essentially for backup and data analytics for global insight extraction in centralized data centers. Data is first ingested at high rates through dedicated systems (*e.g.,* Apache Kafka [30]) and analyzed by Big Data processing frameworks (*e.g.,* Spark [31]). They perform stream and batch analytics on vast historical data, AI model training, and complex simulations. The goal is to help understand *"why"* the phenomena sensed at the Edge are happening.

2) ***Federated Learning Training Use Case****:* To illustrate an Edge-to-Cloud application workflow, we refer to Federated Learning model training. Federated Learning [32] is a collaborative machine learning paradigm that trains a centralized model on decentralized and private data.

The Federated Learning architecture is composed of a central server (typically deployed on the Cloud) and various devices (deployed on the Edge). Edge devices first download a global model from the cloud server and train it for several epochs with their local data. After multiple rounds of model updates, the results are sent to the cloud server for global model aggregation. This training loop continues until the global model achieves the desired accuracy [33].

Capturing provenance data of Federated Learning model training at runtime helps scientists to track model training inputs (*e.g.,* hyperparameters), outputs (*e.g.,* accuracy), and processing history (*e.g.,* training epochs). In this context, captured data from each training epoch may refer to the hyperparameters (input data) followed by the respective accuracy obtained from the training (output data). The goal is to allow users to answer queries like the ones presented in Section I). Analyzing hyperparameters along the model training allows for adapting the training data and fine-tuning the model. Provenance data traces also help in the interpretation and reproducibility of the training results [14], [34].

C. E2Clab: Reproducible Edge-to-Cloud Experiments

E2Clab [35] is an open-source framework (available at [2]) that allows researchers to reproduce the application behavior in a controlled environment to understand and to optimize performance [9]. It sits on top of EnOSlib [36] and implements a rigorous methodology (illustrated in Figure 2) for designing

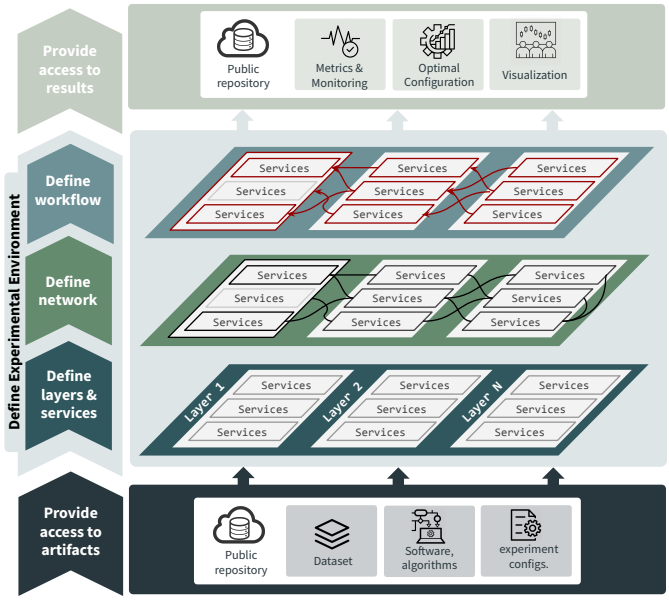


Fig. 2: E2Clab experiment methodology [35].

experiments with real-world workloads on the Edge-to-Cloud Computing Continuum. Section V details how we extend E2Clab to enable provenance data capture of Edge-to-Cloud workflows. Figure 4 illustrates the extended architecture.

High-level features provided by E2Clab are *(i)* reproducible experiments; *(ii)* mapping application parts (executed on Edge, Fog, and Cloud/HPC) and physical testbeds; *(iii)* experiment variation and transparent scaling of scenarios; *(iv)* defining Edge-to-Cloud network constraints; *(v)* automatic experiment deployment, execution, and monitoring (*e.g.,* on various testbeds like Grid'5000 [24], Chameleon [37], and FIT IoT LAB [23]); *(vi)* workflow optimization.

III. Provenance Systems in IoT/Edge

In the literature, to the best of our knowledge, we have not found provenance data capture tools tailored for IoT/Edge devices. Capturing provenance data in such devices requires using tools designed for Cloud/HPC resources. Therefore, in this section, we assess their overhead for capturing data in resource-limited computing resources.

A. Experimental Setup

a) ***Selected provenance systems.****:* Due to the limitations of the PROV-IO and Komadu systems, shown in Table IV, they were excluded from our performance analysis. We choose ProvLake and DfAnalyzer because we have access to their data capture components as open-source software. Since we are limited to testing with the open-source version of these systems, we cannot experiment with features that might deliver lower overhead but are not open-source. For instance, ProvLake reports being able to use a different communication protocol other than HTTP 1.1 for machine learning provenance capture with low overhead in an HPC environment [14], but this system version is not available as open-source.

TABLE I: Synthetic workload configurations.

Configurations to generate the synthetic workloads				
Number of chained transformations	5			
Number of tasks	100			
Attributes per task	10	100		
Task duration (s)	0.5	1	3.5	5

*b) **Performance metrics.**:* The main analyzed metric is the *capture time overhead*, which refers to the relative difference of the workflow execution time with and without data capture. We repeat the experiment 10 times for each provenance system and for each synthetic workload and report the mean followed by the 95% confidence interval.

*c) **Overhead levels.**:* In the literature, the reference to *low overhead* or *negligible overhead*, in terms of provenance capture time in Cloud/HPC environments, differs between application domains. For instance: $<2\%$ for blockchains [38]; $\leqslant 4\%$ for I/O-centric workflows [21]; 4% for AI model training [13]; $\leqslant 12\%$ for security applications [39]; to cite a few. Regarding provenance capture on resource-limited IoT/Edge devices, prohibitive overhead levels may vary depending on the application use case. For instance, in latency-sensitive applications such as autonomous vehicles [40], real-time monitoring in smart energy grids [41], and virtual and augmented reality [42], to cite a few, a $> 3\%$ processing time overhead is considered high (*i.e.,* enough to exceed the acceptable latency thresholds) as it can introduce delays that disrupt the real-time nature of the application, leading to inaccuracies, missed targets, or compromised safety.

*d) **Synthetic workload.**:* We use a synthetic workload to evaluate the provenance capture overhead because doing it in real workloads is much more complicated, costly, and may not make sense for the real application. The reason is that we cannot precisely control and isolate variables such as elapsed time, number of tasks, and number of attributes. A similar situation happens when scientists need to rely on simulations instead of real phenomena to test and evaluate their hypotheses. Unfortunately, there are no well-established benchmarks in the community to evaluate overhead in provenance systems. Therefore, like related work [17], [18], we decided to focus our analysis on synthetic workload configurations. Such configurations are based on real-life workloads [43]–[45], and we refined the configuration space of our workloads with preliminary experiments on real-life edge devices.

Table I presents the 8 synthetic workload configurations used to analyze the data capture overhead. We chose these values to cover combinations of application characteristics. The idea of these configurations is to mimic the characteristics of the various real-life workloads that IoT/Edge devices typically execute, such as AI model training (*e.g.,* the Federated Learning use case we presented earlier), image pre-processing, and sensor data aggregation, among others. Such workloads are composed of various tasks (number of tasks), each one with a different number of attributes (attributes per task) and with different processing times (task duration).

We consider workloads with 5 chained transformations, which is an approximate number of transformations in many applications. In the Federated Learning application, for example, one of the transformations is model training, which has many epoch executions. We consider each epoch execution as a task of the model training transformation and each epoch has associated features (considered input attributes) and performance metrics (considered output attributes) [14]. Other transformations include data preparation and the evaluation of the trained model. To generate our synthetic workload, we consider 100 tasks. In the Federated Learning example, it would represent a training with 100 epochs. For each task, we represent applications that manipulate a few (about 10) or more (about 100) attributes per task. Besides, to represent various classes of applications, we also consider four different task duration: shorter (*e.g.,* 0.5 or 1 seconds) and longer (*e.g.,* 3.5 or 5 seconds).

We run **preliminary experiments to refine** the synthetic workload configurations. We observe that there is no significant impact on the capture overhead when varying the *number of tasks* from 10, 50 to 100. In addition, since the data capture and transmission is measured per task, **mainly variations in the number of** *attributes per task* (amount of data transmitted) and *task duration* (data capture frequency) impact the capture time overhead (calculated as the relative difference).

*e) **Hardware.**:* Each workload configuration runs on a single A8-M3 [46] IoT device (ARM Cortex-A8 microprocessor, 600Mhz, 256MB; radio: 802.15.4, 2.4 GHz; power: 3.7V LiPo battery, 650 mAh) available at the FIT IoT LAB testbed [23]. We instrument the synthetic workloads (code available at [47]) with the capture libraries provided by ProvLake and DfAnalyzer systems. The libraries transmit the data to the provenance system running on a remote Cloud/HPC server [48] (Intel Xeon Gold 5220, 2.20GHz, 18 cores; 96GB RAM; Ethernet) available at the Grid'5000 [24] testbed.

B. Overhead Analysis

Table II presents the **capture time overhead** of ProvLake and DfAnalyzer **in IoT/Edge devices**, and Table III shows the analysis of a feature provided by ProvLake, which consists of **grouping the captured data**, *i.e.,* messages, before transmitting them to the server, *i.e.,* provenance system. In addition, we analyze how low-bandwidth networks may impact such data grouping strategy.

Results in Table II show that both systems present high overhead ($>39\%$) for tasks with a duration of 0.5 seconds. For the remaining task duration, the overhead is still high ($>3\%$). Varying the number of attributes per task from 10 to 100 slightly increases the overhead.

Regarding Table III, we observe low overhead ($<3\%$) when grouping 50 messages for a task duration of 0.5 seconds, and grouping from 20 messages for a task duration of 1 second, for 1Gbit bandwidth. While for 25Kbit bandwidth, we observe high overhead ($>43\%$) for all workloads.

TABLE II: Capture overhead of ProvLake and DfAnalyzer.

overhead level	low ⩽ 3%	high >3%

attributes per task	Provenance System	Capture Overhead (%)			
10	ProvLake	56.9% ±0.08	29.9% ±0.29	8.56% ±0.01	6.02% ±0.01
10	DfAnalyzer	39.8% ±0.06	21.2% ±0.34	6.12% ±0.07	4.26% ±0.01
100	ProvLake	57.3% ±0.10	30.1% ±0.41	8.57% ±0.01	6.04% ±0.04
100	DfAnalyzer	40.5% ±0.20	21.3% ±0.06	6.12% ±0.01	4.31% ±0.01
	task dur. (s)	0.5	1	3.5	5

TABLE III: ProvLake: impact of bandwidth and grouping strategy on the capture overhead.

# of messages grouped	Bandwidth 1Gbit		Bandwidth 25Kbit	
0	57.3% ±0.10	30.1% ±0.27	321% ±1.05	161% ±1.14
10	6.83% ±0.02	3.58% ±0.20	102.5% ±3.89	49.8% ±2.92
20	3.87% ±0.01	1.99% ±0.01	100.8% ±3.78	51.16% ±1.03
50	2.37% ±0.01	1.24% ±0.01	95.04% ±0.10	43.23% ±0.28
task duration (s)	0.5	1	0.5	1

C. Design-level Limitations of Existing Systems

Table IV presents the takeaways of our performance analysis and exposes the main limitations of the existing provenance systems. In summary, the evaluation shows that the existing systems present high overheads (>3%) when capturing on IoT/Edge devices.

ProvLake and DfAnalyzer rely on HTTP over TCP, instead of IoT-based messaging and transmission protocols such as MQTT [49], CoAP [50], AMQP [51], UDP [52], RPL [53], to cite a few. In resource-constrained devices, they make a relevant impact on performance, resource usage, and power consumption, as explored by existing works [54]–[56].

The experiment results reinforce the need for capture approaches tailored to the constraints imposed by IoT devices. In addition, simplified data models to represent the provenance data help to reduce overheads.

IV. ProvLight Design

This section introduces ProvLight, a tool [1] for the efficient provenance data capture of Edge-to-Cloud workflows. ProvLight is designed to capture provenance in IoT/Edge devices with low overhead in terms of capture time, CPU and memory usage, network usage, and power consumption.

Subsection IV-A presents the ProvLight provenance model. Next, the architectural details are given in Subsection IV-B, while Subsection IV-C describes its implementation.

TABLE IV: Limitations of existing provenance systems.

System	Limitation
DfAnalyzer	Presents **high** (>3%) **capture overhead** for all synthetic workloads.
ProvLake	Presents **high** (>3%) **overhead** for all workloads. However, ProvLake allows **grouping captured data** to reduce transmission frequency, enabling lower overhead, but it **still suffers high overhead** in **low bandwidth** networks.
PROV-IO	Does not send the captured data over the network to another machine hosting the provenance system. Instead, it periodically **dumps the in-memory provenance graph to disk**. This approach is **not suitable** for IoT/Edge devices.
Komadu	Komadu does not follow a clear separation between a client library and a backend provenance server. Therefore, the **capture** and the **processing** of the captured information **run in the same machine**. This approach is **not suitable** for capturing on IoT/Edge devices.

A. Data Exchange Model

ProvLight provenance data exchange model follows the W3C PROV-DM [28] recommendation. The goal is to have a data exchange schema (domain-agnostic PROV modeling) for capturing data in the IoT/Edge and making sure these captured data are compatible with W3C PROV-based workflow provenance systems, such as ProvLake, DfAnalyzer, PROV-IO, among many others. Table V describes ProvLight classes and their relationships and maps them to PROV-DM core elements.

The main classes of our model are *Workflow*, *Task*, and *Data*. These classes are derived from the *Agent*, *Activity*, and *Entity* PROV-DM types, respectively. ProvLight classes aim to provide a simplified abstraction allowing users to track workflow (*Workflow* class), input and output parameters (*Data* class), and processing history (*Task* class).

The *Workflow* class may be used to refer to the application workflow (*e.g.,* Federated Learning training). The *Task* class refers to the tasks executed in the workflow (*e.g.,* each epoch or model update of the model training). Finally, the *Data* class represents the input data attributes and values (*e.g.,* hyperparameters of the learning algorithm) or the output attributes (*e.g.,* training time and loss of each epoch).

To represent PROV-DM relationships, we use the *id* attribute of each class. We link the *Task* and *Data* classes with the workflow they belong to (*wasAssociatedWith* and *wasAttributedTo*, respectively). The links between a *Task* and its respective *Data* inputs and the generated outputs are represented by the *used* and *wasGeneratedBy* relationships, respectively. The *dependencies* attribute in the *Task* class links tasks (*wasInformedBy*) with dependencies (*e.g.,* task B starts after task A ends). Finally, the *derivations* attribute in the *Data* class links (*wasDerivedFrom*) chained data (*e.g.,* data D_A was used in task A to generate data D_B).

TABLE V: The ProvLight provenance data exchange model follows PROV-DM.

PROV-DM Type	ProvLight Class	ProvLight Class Attributes	ProvLight Attribute Description and PROV-DM Relationships	ProvLight Class Description
Agent	Workflow	id	Workflow id.	Refers to application workflows.
Activity	Task	id	Task id.	Represents the processing steps of tasks (and their dependencies) that compose workflows.
		workflow	Links tasks with the workflow they belong to (*wasAssociatedWith*).	
		dependencies	Dependencies between tasks (*wasInformedBy*).	
		data	Data used (*used*) and generated (*wasGeneratedBy*) by a task.	
		time	Task start and end time.	
		status	Task status: running or finished.	
Entity	Data	id	Data id.	Represents data derivations along the workflow execution.
		workflow_id	Links data with the workflow they belong to (*wasAttributedTo*).	
		derivations	Links chained data (*wasDerivedFrom*).	
		attributes	Data attributes and values.	

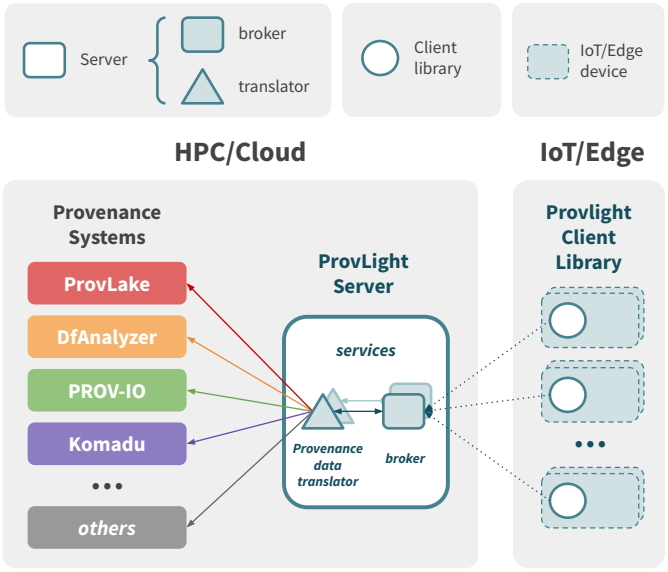


Fig. 3: ProvLight Architecture.

TABLE VI: How does ProvLight differ from state-of-the-art systems in terms of data capture?

	ProvLight	DfAnalyzer	ProvLake
application layer protocol	MQTT-SN *QoS 2: Exactly once*	HTTP 1.1	HTTP 1.1
transport layer protocol	UDP	TCP	TCP
Communication model	Publish/ Subscribe	Request/ Response	Request/ Response
Server side	MQTT-SN Broker	HTTP Server	HTTP Server
Client side features	provenance data representation & payload compression & grouping data captured	N/A	grouping data captured
Provenance data model	PROV-DM	PROV-DM	PROV-DM
Capture library language	Python	Python, C++	Python

Defining such relations aims to provide users with the data processing history: *Where* did the data come from? *How* was the data transformed? and *Who* acted upon it? For instance, capturing provenance data of Federated Learning model training workflows may help users to interpret results. Tracking model training at runtime and fine-tuning hyperparameters is helpful, especially when the training process takes a long time.

B. Architecture

Figure 3 presents the ProvLight architecture. It follows a *client/server* model where the *server* receives the captured data from *clients* and then translates it and sends it to provenance systems. We highlight that ProvLight may integrate with existing provenance systems like DfAnalyzer, ProvLake, and PROV-IO, among others (*e.g.,* through their APIs and ProvLight data translator), as a solution for capturing data of workflows running on IoT/Edge devices, as illustrated in Figure 3. Table VI summarizes how the ProvLight architecture design differs from the systems analyzed in Section III.

This integration may be achieved by using:

1) ***Server****:* The ProvLight *server* is composed of a *broker* and a *provenance data translator*. Both may be parallelized to scale the data capture for scenarios with various IoT/Edge devices. We describe the main roles of each one.

(i) Broker: refers to an MQTT-SN broker (MQTT for Sensor Networks [57]). During workflow execution, *clients* subscribe to the *broker* and then start to transmit the captured data. Next, this data is forwarded to the *provenance data translator*, which is subscribed to the *broker*.

(ii) Provenance Data Translator: translates the captured data to the respective format used by the provenance system. The *provenance data translator* may be extended, by users, to translate to a particular data model of a provenance system. After translating, it sends the data to the provenance system service (*e.g.,* typically available at an *ip:port*). It allows seamless integration with existing systems.

2) ***Client****:* The ProvLight *client* aims to efficiently capture provenance data on resource-limited devices. ProvLight provides a client library that follows the W3C PROV-DM provenance model (as presented in Table V). This library

```
1  from provlight import Workflow, Task, Data
2  attributes = 100
3  chained_transformations = 5
4  number_of_tasks = 100
5  # Application Workflow
6  workflow = Workflow(1)
7  workflow.begin()
8  # Tasks and data derivations
9  data_id = 0
10 previous_task = []
11 in_data = {'in': [1 for _ in range(attributes)]}
12 out_data = {'out': [2 for _ in range(attributes)]}
13 for transf_id in range(chained_transformations):
14     for task_id in range(int(number_of_tasks/
       chained_transformations)):
15         data_id += 1
16         task = Task(transf_id-task_id, workflow, transf_id,
         dependencies=previous_task)
17         data_in= Data(in_{data_id}, workflow.id, in_data)
18         task.begin([data_in])
19         #### ADD YOUR TASK HERE ####
20         data_out= Data(out_{data_id}, workflow.id,out_data)
21         task.end([data_out])
22         previous_task = [task.id]
23 workflow.end()
```

Listing 1: ProvLight: user-defined provenance capture.

allows users to instrument their workflow code to decide what data to capture. A *client* is configured to transmit, at runtime, the captured data to the remote *broker* (*e.g., ip:port*). This allows users to track workflow execution at runtime (*e.g.,* started and finished tasks, input and output data, *etc.*) through provenance systems supporting data ingestion at runtime.

C. Implementation

1) ***Server****:* The ***Broker*** is implemented based on the Eclipse RSMB server [58] (Really Small Message Broker). RSMB builds on top of Mosquitto [59] and implements the MQTT-SN protocol.

The ***Provenance Data Translator*** is a Python service that may be extended to translate captured data (from the ProvLight data format) to a particular provenance system (*e.g.,* DfAnalyzer, ProvLake, Komadu, *etc.*). In our repository [1], we provide an implementation showing how to translate from the ProvLight data format to DfAnalyzer. Such translation is possible since the aforementioned systems follow the W3C PROV-DM provenance model. For the *translator-to-broker* communication, we use the MQTT-SN Python client library [60] based on Eclipse RSMB. Finally, for the *translator-to-provenance-system* communication, users are free to use any Python library compatible with the provenance system (*e.g.,* Requests [61]).

2) ***Client****:* The ProvLight client library is implemented in Python and provides a series of features targeting resource-limited IoT/Edge devices:

- *provenance data representation:* simplified classes for provenance modeling that allow users to represent workflows, data derivations (*e.g.,* input/output data from tasks) and tasks (*e.g.,* status, dependencies, data derivations);
- *payload compression:* compresses the bytes in captured data before transmitting over the network; and
- *data capture grouping:* allow users to optionally group data just from ended tasks, so users may still track at workflow runtime the tasks that have already started.

As shown later in the evaluation section, grouping and compressing captured data help reduce capture time overhead, especially in IoT/Edge devices.

How to capture provenance data from the workflows? Listing 1 illustrates an example of application code instrumentation with the ProvLight library highlighted in blue color. Lines 6, 7, and 23 instantiate the workflow, start, and finalize it, respectively. Line 16 instantiates a task, linking it to the *workflow*, input data *derivation*, and dependent task. Lines 18 and 21 capture data from the initialization and finalization of the task. Before starting a task, line 17 instantiates *Data* and adds it as input data (line 18) to the task. Following the same logic, line 20 instantiates and adds the output data from the task. We highlight that the *begin()* and *end()* methods of *Workflow* and *Task* transmit the captured data over the network to the *broker*. Finally, line 19 is where the workflow task runs.

V. Provenance Capture of Edge-to-Cloud Workflows

This section presents the integration of ProvLight as a key system in the E2Clab [35] framework for reproducible experimentation across the Edge-to-Cloud Continuum. This integration allows users to capture end-to-end provenance data of Edge-to-Cloud workflows. Figure 4 shows the extended E2Clab architecture with the new components in the red color.

A. Provenance Manager

We design a new manager named *Provenance Manager*. Figure 4 illustrates the integrated view of the two main elements that compose the Provenance Manager:

(i) ProvLight: to efficiently capture provenance data of workflows running on IoT devices. It also allows users to capture provenance in Cloud/HPC environments. ProvLight translates the captured data to the DfAnalyzer data model.

(ii) DfAnalyzer: to store and query provenance captured by *ProvLight* during workflow runtime (*e.g.,* compare provenance of multiple workflow evaluations to understand how they impact on performance). Furthermore, it allows users to visualize dataflow specifications (*i.e.,* data attributes of each dataset).

In addition to the characteristics of the provenance systems analyzed in Table IV, and due to ProvLake being proprietary within IBM, while DfAnalyzer is open source [62], in this work we decide to use DfAnalyzer. As the data capture component of DfAnalyzer presents high overhead, we just use its data analysis and storage components. Finally, the *Provenance Manager* could replace DfAnalyzer with other provenance systems (*e.g.,* PROV-IO, Komadu, *etc.*). It requires extending ProvLight to translate the provenance data to the data model of the provenance system and using their APIs.

B. Provenance Capture

Through the E2Clab framework, users may easily enable provenance data capture across the Edge-to-Cloud continuum

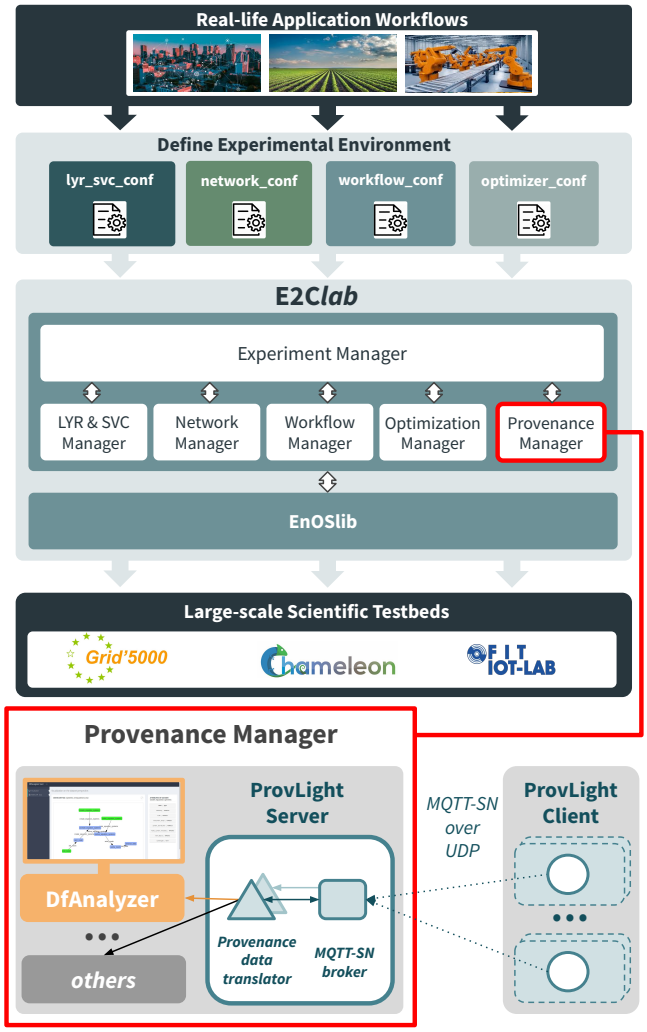


Fig. 4: Extended E2Clab: Provenance Data Manager.

```
1 environment:
2   g5k: cluster: gros
3   iotlab: cluster: grenoble
4 provenance: ProvenanceManager
5 layers:
6 - name: cloud
7   services:
8   - name: Server, environment: g5k, qtd: 1
9 - name: edge
10  services:
11  - name: Client, environment: iotlab, arch: a8, qtd: 64
```

Listing 2: E2Clab: provenance of Edge-to-Cloud workflows.

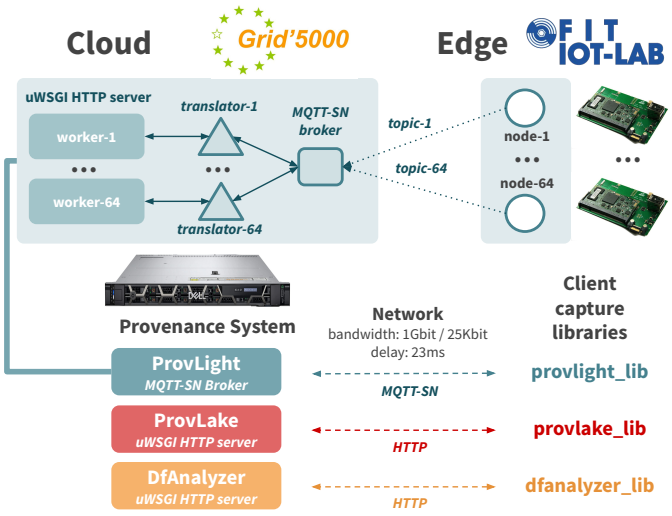


Fig. 5: Experimental setup.

through simple configuration files, as illustrated in Listing 2. Listing 2 refers to the E2Clab *layers_services.yaml* configuration file used to setup the experimental environment (*e.g.,* testbeds, services that compose workflows, *etc.*). Lines 2 and 3 request resources from Grid'5000 and FIT IoT LAB testbeds, respectively. Line 8 requests a single server (*e.g.,* Federated Learning server) on the Cloud layer; while line 11 requests 64 clients (*e.g.,* to train the model with their local data) on the Edge layer. Finally, line 4 setups the provenance data capture (the *ProvenanceManager* service). After that, users may instrument their application code to capture data, as presented in Listing 1.

The *ProvenanceManager* service starts a Docker [63] container with the DfAnalyzer provenance system and a ProvLight container allowing clients to send their provenance data. DfAnalyze exhibits at workflow runtime the captured data on its Web interface. The *ProvenanceManager* service may be easily plugged into other provenance systems by just using their Docker images and extending the provenance data translator.

VI. Evaluation

We aim to answer the following research questions: ***how does ProvLight perform in IoT/Edge devices?*** *while initially targeting resource-constrained Edge devices,* ***can ProvLight be efficiently used also in the Cloud?*** We answer these questions in subsections A—D and E, respectively, by comparing ProvLight against ProvLake and DfAnalyzer.

The *main performance metric* is the capture overhead in terms of: *(i)* data capture time; *(ii)* CPU and memory usage; *(iii)* network usage; and *(iv)* power consumption. **The experimental setup is the same as presented in Subsection III-A**, with synthetic workloads generated based on the Federated Learning use case. The deployment is shown in Figure 5. Results in Figure 6 are the mean of 10 runs with their 95% confidence interval.

A. Capture Time Overhead

Table VII presents the capture time overhead comparison for the 8 synthetic workloads. **In summary, ProvLight presents low capture overhead (<3%) for all workloads analyzed.** Regarding tasks with a duration of 3.5 seconds or more, the capture overhead of ProvLight is below 0.5%. Varying the number of attributes per task from 10 to 100 does not significantly increase the capture time. We highlight that **ProvLight is about 37x and 26x faster than ProvLake and DfAnalyzer, respectively.**

TABLE VII: ProvLight: capture overhead in IoT/Edge devices. Refer to Table II to compare with DfAnalyzer and ProvLake.

Attributes per task	Capture Overhead (%)			
10	1.45% ±0.01	1.02% ±0.01	0.31% ±0.01	0.23% ±0.01
100	1.54% ±0.01	1.11% ±0.01	0.37% ±0.01	0.29% ±0.01
task duration (s)	**0.5**	**1**	**3.5**	**5**

TABLE VIII: ProvLight: impact of bandwidth and grouping strategy on the capture overhead. Refer to Table III to compare with ProvLake.

# of messages grouped	Bandwidth 1Gbit		Bandwidth 25Kbit	
0	1.54% ±0.01	1.10% ±0.01	1.56% ±0.01	1.04% ±0.01
10	1.37% ±0.01	0.75% ±0.01	1.37% ±0.01	0.74% ±0.01
20	1.32% ±0.01	0.72% ±0.01	1.34% ±0.01	0.73% ±0.01
50	1.31% ±0.01	0.72% ±0.01	1.31% ±0.01	0.72% ±0.01
task duration (s)	**0.5**	**1**	**0.5**	**1**

Similarly to Table III, Table VIII zooms our analysis in order to understand the impact of bandwidth variations and the grouping strategy on the data capture time. Results show that, differently from ProvLake, **ProvLight presents low capture time overhead in low-bandwidth scenarios for task durations of 0.5 and 1 second.** We highlight that, especially in low-bandwidth scenarios (25Kbit), the ProvLight grouping strategy presents low overhead (<2%), while ProvLake presents high overhead (>43%), see Table III.

Scalability analysis. Table IX presents the capture time overhead of ProvLight when scaling the number of IoT/Edge devices and considering 100 tasks of 0.5s each and 100 attributes per task. We scale the scenario with 8, 16, 32, and 64 devices capturing provenance data in parallel and sending the data to the cloud server. As illustrated in Figure 5, each client sends its data to its respective topic in the *Broker* and we parallelized the number of *translators* accordingly. Lastly, provenance systems (*i.e.,* DfAnalyzer in our case) can handle parallel requests and store the provenance data in a database system (*e.g.,* MonetDB [64] used in DfAnalyzer). Results show that **by scaling up to 64 devices, the capture overhead is low (<3%)** and does not significantly impact the capture time. This is expected because devices (clients) asynchronously publish their messages to their respective topics in the *MQTT-SN Broker*. For 8 and 64 devices, the capture time overhead is 1.54% and 1.57%, respectively.

B. CPU and Memory Overhead

Figures 6a and 6b present the CPU and memory overhead for capturing provenance data with ProvLake, DfAnalyzer, and ProvLight (from left to right). Regarding the CPU overhead,

TABLE IX: ProvLight scalability analysis.

System	Capture Overhead (%)			
ProvLight	1.54% ±0.01	1.54% ±0.01	1.56% ±0.01	1.57% ±0.02
# of devices	**8**	**16**	**32**	**64**

TABLE X: Capture overhead in Cloud servers.

	System	Capture Overhead (%)			
100 attributes per task	ProvLake	1.71% ±0.03	0.92% ±0.01	0.34% ±0.01	0.26% ±0.01
	DfAnalyzer	1.17% ±0.02	0.63% ±0.01	0.25% ±0.01	0.21% ±0.01
	ProvLight	0.24% ±0.01	0.17% ±0.01	0.12% ±0.01	0.11% ±0.01
	task duration (s)	**0.5**	**1**	**3.5**	**5**

ProvLight uses 7x and 5x less CPU than ProvLake and DfAnalyzer, respectively. Capturing with ProvLight, the CPU overhead is low (<3%), and CPU usage varies between 1.7% and 2%. Regarding the memory overhead, ProvLight memory usage is <4%. It uses about 2x less memory than ProvLake and DfAnalyzer.

C. Network Usage Overhead

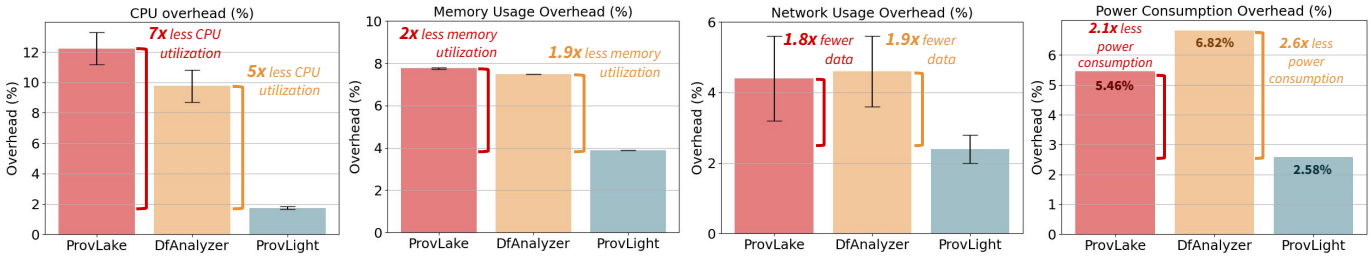
As presented in Figure 6c, **ProvLight transmits about 2x less data than ProvLake and DfAnalyzer.** ProvLight network usage is around 3.7 KB/sec during data capture. The application layer protocol used in ProvLight (*e.g.,* MQTT-SN), which compresses captured data before transmitting it, especially for tasks with many attributes per task (*e.g.,* 100 in this case), explains such difference (2x less data) when compared to the other capture approaches.

D. Power Consumption Overhead

Finally, results in Figure 6d (error bar omitted because we use the maximum power consumption for capturing provenance data) show that ProvLight power consumption overhead is 2.1x and 2.6x less than ProvLake and Dfanalyzer. We highlight that **ProvLight overhead is 2.58% (considered low, <3%), against 5.46% (ProvLake) and 6.8% (DfAnalyzer).** The power consumption (in watts) for capturing and transmitting the data is on average 1.43W, 1.47W, and 1.49W for ProvLight, ProvLake, and DfAnalyzer, respectively.

E. Performance in Cloud Servers

We compare the capture time overhead of ProvLight against ProvLake and DfAnalyzer in Cloud servers (*i.e.,* data capture on a server [48] available in Grid'5000). Experiment results in Table X show that the three approaches present low capture overhead (<3%) for all task durations. Similarly to IoT/Edge devices, ProvLight also outperforms ProvLake and DfAnalyzer in Cloud servers. ProvLight is 7x and 5x faster than ProvLake and DfAnalyzer, respectively. ProvLight capture time overhead is very low (<0.25%) for all task durations.



(a) CPU overhead. (b) Memory overhead. (c) Network overhead. (d) Power overhead.

Fig. 6: Provenance data capture overhead with respect to: CPU, memory, network usage, and power consumption.

VII. Discussion

The integration of ProvLight as a key system within the E2Clab framework exhibits a series of features that make E2Clab a promising platform for future performance optimization of applications on the Edge-to-Cloud Continuum through efficient provenance capture and reproducible experiments.

A. ProvLight Design Choices Impact on Performance

As presented in Table VI, the combination of ProvLight design choices on the server and client sides contributed to the low capture overhead. The ProvLight client library keeps the connection to the remote server open while capturing data (*i.e.,* when capturing data from different tasks, the connection is reused). Additionally, the library is based on the publish/-subscribe asynchronous communication model and it uses MQTT-SN (application layer protocol) over UDP (transport layer protocol) instead of HTTP over TCP. Despite TCP being more reliable (*e.g.,* uses acknowledgment messages for data delivery), the ProvLight client sends data using QoS level 2, which guarantees that each message is received exactly once by the recipient. Such design choices help to reduce connection overheads while data transmission handshakes/acknowledgments require less bandwidth.

Another important feature is that ProvLight compresses data (using binary format) before transmitting. Through preliminary experiments, we analyzed the performance trade-offs of compressing the data on the IoT/Edge devices to make sure it is worth adding that feature. The time required to compress data (*e.g.,* tasks with 100 attributes) on the edge device is negligible, around 0.001s on average.

Our analysis considered low-bandwidth scenarios and also the data grouping strategy, resulting in fewer and larger messages to reduce the number of transmissions. We also observe that the overhead of decompressing and translating such data on the Cloud server is negligible, around 0.005s.

Data communication is key to performance efficiency in IoT/Edge workloads, especially for low bandwidth networks. ProvLight design choices such as simplified capture library for provenance data exchange (see Table V), asynchronous MQTT-SN over UDP, data grouping, and data compression, explain the positive effects on performance and costs (*e.g.,* lower overheads in terms of data capture time, and CPU, memory, network usage, and energy consumption).

In summary, the lightweight asynchronous protocol (MQTT-SN over UDP) has a major impact on the capture time overhead, energy consumption, and CPU and network usage. Our simplified data model has a major impact on memory consumption, and it helps to reduce even more the capture time overhead and CPU usage by 1.7% and 1.4%, respectively.

B. Impact of ProvLight on Real-life Use-Cases

To illustrate how real-life use cases could benefit from ProvLight and its integration in the E2Clab framework, we consider the training of Neural Networks presented in [15] and [13]. In these articles, the authors use the storage and query components of DfAnalyzer to store captured data during model training executed on Cloud/HPC infrastructure and then query the data. They demonstrate how provenance data may be used to answer queries like the ones we presented in Section I.

Since modern AI workflows are being **executed on hybrid infrastructures**, we may instantiate this use-case (Neural Network training on the Cloud/HPC) to the context of hybrid Edge-to-Cloud Federated Learning Neural Network training. In this hybrid context, the model is now trained on various resource-limited Edge devices. Thanks to the efficient capture approach of ProvLight, users may still track the model training by capturing provenance data. Without ProvLight, capturing provenance data of this use-case on the IoT/Edge is **prohibitive due to the high overheads** imposed by the existing approaches, as presented in Section III.

Finally, thanks to the E2Clab framework, users may easily set up the Federated Learning Neural Network training and deploy it on distributed Edge devices (to train the model) and on the Cloud server (to update the global model). Furthermore, the E2Clab *Provenance Manager* allows users to store data captured with ProvLight and query them using DfAnalyzer. Therefore, through the E2Clab *Provenance Manager*, users may answer the same queries mentioned earlier. We highlight that this Neural Network use case is just one example from various that could benefit from this work.

C. Integration with Existing Systems

ProvLight is designed to be easily integrated with existing provenance systems (*e.g.,* ProvLake, DfAnalyzer, PROV-IO, among others) and workflow management systems and deployment frameworks (*e.g.,* Pegasus, E2Clab, among others). Such

integration would enable these systems to capture provenance data (with low capture overheads) in IoT/Edge devices.

As presented in Subsection IV-B, this is possible thanks to the ProvLight *provenance data translator*. It translates from the ProvLight data format to the data format of the target system. This requires users to extend the ProvLight translator. In this work, we demonstrate in Section V: *(i)* the integration of ProvLight with the open-source DfAnalyzer provenance system as a solution for provenance capture on the IoT/Edge; and then *(ii)* we integrate this capture solution within the E2Clab framework (the *Provenance Manager*) to enable provenance capture of Edge-to-Cloud workflows.

D. Reproducibility and Artifact Availability

The experimental evaluations presented in this work follow a rigorous methodology [35] to support reproducible Edge-to-Cloud experiments on large-scale testbeds (*e.g.,* Grid'5000 and FIT IoT LAB used in our experiments). This guided us to systematically define the experimental environment (*e.g.,* computing resources, services/systems, network, and application execution) through well-structured configuration files. The experiment artifacts and results are available at [47].

VIII. Related Work

Tanaka et al. [8] extend the Pegasus [65] Workflow Management System to support Edge-to-Cloud workflows. The paper explores performance trade-offs in managing and executing Edge-to-Cloud workloads. Pegasus provides provenance data collection capabilities to capture performance metrics during workflow execution. We highlight that Pegasus (and other systems like Kepler [66], Taverna [67], *etc.*) explores the **predefined** provenance capture approach. Pegasus automatically logs provenance data about the local execution of the application codes, such as launching them and capturing the exit status and runtime information [68]. While ProvLight and the systems we compared with (see Table IV) explore the **user-defined** capture approach, *i.e.,* the user defines what to capture by workflow code instrumentation. Furthermore, unlike ProvLight, Pegasus does not explore IoT/Edge protocols to transfer the captured data nor provides features like simplified data models and compressing and grouping messages. This may result in higher overheads compared to ProvLight, as presented in Section III. Finally, the authors do not analyze the energy consumption of their capture approach.

A provenance collection framework for the IoT/Edge devices is proposed in [69]. The proposed framework follows PROV-DM recommendations and provides provenance collection capabilities for IoT/Edge devices. Unlike our work, the authors do not validate their approach on real-life Edge devices. Also, no performance evaluations are presented to understand capture overheads.

Genoma, a distributed provenance-as-a-service system across IoT/Edge devices and Cloud servers, is proposed in [70]. Genoma transmits provenance data to the Cloud using the MQTT protocol. Data is transmitted based on storage availability on the Edge device and the frequency of data communication. The authors do not evaluate the performance of Genoma. Capture overheads regarding capture time, network usage, energy consumption, and CPU and memory usage are left for future work. In contrast, ProvLight is evaluated on all the metrics mentioned above.

IX. Conclusion

The integration of ProvLight within E2Clab makes the latter, to the best of our knowledge, the first framework to support the end-to-end provenance capture of Edge-to-Cloud workflows with low overheads across the Computing Continuum. ProvLight and E2Clab are available as open-source tools. In future work, we will enable the provenance capture of workflows developed in C/C++ (not only in Python) and secure the data transmission from the Edge devices to the provenance system.

Acknowledgments

This work was funded by Inria through the HPC-BigData Inria Challenge (IPL), by the French ANR OverFlow project (ANR-15- CE25-0003), and the HPDaSc associate team with Brazil. Marta Mattoso and Débora Pina are funded by CNPq and FAPERJ. Renan is at the Oak Ridge Leadership Computing Facility at the Oak Ridge National Laboratory, which is supported by the Office of Science of the U.S. Department of Energy under Contract No. DE-AC05-00OR22725. Experiments presented in this paper were carried out using the Grid'5000 and FIT IoT LAB testbeds, supported by a scientific interest group hosted by several Universities and organizations.

References

[1] (2023, jan) Provlight tool: Provenance capture for iot/edge devices. [Online]. Available: https://gitlab.inria.fr/provlight/provlight

[2] (2019, jan) E2clab source code. [Online]. Available: https://gitlab.inria.fr/E2Clab/e2clab

[3] D. Rosendo, A. Costan, P. Valduriez, and G. Antoniu, "Distributed intelligence on the Edge-to-Cloud Continuum: A systematic literature review," *Journal of Parallel and Distributed Computing*, vol. 166, pp. 71–94, Aug. 2022. [Online]. Available: https://hal.archives-ouvertes.fr/hal-03654722

[4] S. Deng, H. Zhao, W. Fang, J. Yin, S. Dustdar, and A. Y. Zomaya, "Edge intelligence: The confluence of edge computing and artificial intelligence," *IEEE Internet of Things Journal*, vol. 7, no. 8, pp. 7457–7469, 2020.

[5] M. Asch, T. Moore, R. Badia, M. Beck, P. Beckman, T. Bidot, F. Bodin, F. Cappello, A. Choudhary, B. de Supinski *et al.*, "Big data and extreme-scale computing: Pathways to convergence - toward a shaping strategy for a future software and data ecosystem for scientific inquiry," *The International Journal of High Performance Computing Applications*, vol. 32, no. 4, pp. 435–479, 2018.

[6] ETP4HPC, "Etp4hpc strategic research agenda," https://www.etp4hpc.eu/sra.html, April 29, 2020.

[7] Y. Xia, X. Etchevers, L. Letondeur, A. Lebre, T. Coupaye, and F. Desprez, "Combining heuristics to optimize and scale the placement of iot applications in the fog," in *2018 IEEE/ACM 11th International Conference on Utility and Cloud Computing (UCC)*. IEEE, 2018, pp. 153–163.

[8] R. Tanaka, G. Papadimitriou, S. C. Viswanath, C. Wang, E. Lyons, K. Thareja, C. Qu, A. Esquivel, E. Deelman, A. Mandal *et al.*, "Automating edge-to-cloud workflows for science: Traversing the edge-to-cloud continuum with pegasus," in *2022 22nd IEEE International Symposium on Cluster, Cloud and Internet Computing (CCGrid)*. IEEE, 2022, pp. 826–833.

[9] D. Rosendo, A. Costan, G. Antoniu, M. Simonin, J.-C. Lombardo, A. Joly, and P. Valduriez, "Reproducible Performance Optimization of Complex Applications on the Edge-to-Cloud Continuum," in *Cluster 2021 - IEEE International Conference on Cluster Computing*, Portland, OR, United States, Sep. 2021, pp. 23–34. [Online]. Available: https://hal.archives-ouvertes.fr/hal-03310540

[10] R. Kumar, M. Baughman, R. Chard, Z. Li, Y. Babuji, I. Foster, and K. Chard, "Coding the computing continuum: Fluid function execution in heterogeneous computing environments," in *2021 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*. IEEE, 2021, pp. 66–75.

[11] L. Liu and M. T. Özsu, *Encyclopedia of database systems*. Springer, 2009, vol. 6.

[12] R. Souza, V. Silva, J. J. Camata, A. L. G. A. Coutinho, P. Valduriez, and M. Mattoso, "Keeping track of user steering actions in dynamic workflows," *Future Generation Computer Systems*, vol. 99, pp. 624–643, 2019.

[13] R. M. Silva, D. Pina, L. Kunstmann, D. de Oliveira, P. Valduriez, A. L. Coutinho, and M. Mattoso, "Capturing provenance to improve the model training of pinns: first handon experiences with grid5000," in *CILAMCE-PANACM 2021-Proceedings of the joint XLII Ibero-Latin-American Congress on Computational Methods in Engineering and III Pan-American Congress on Computational Mechanics*, 2021, pp. 1–7.

[14] R. Souza, L. G. Azevedo, V. Lourenço, E. Soares, R. Thiago, R. Brandão, D. Civitarese, E. Vital Brazil, M. Moreno, P. Valduriez, M. Mattoso, R. Cerqueira, and M. A. S. Netto, "Workflow provenance in the lifecycle of scientific machine learning," *Concurrency and Computation: Practice and Experience*, vol. e6544, pp. 1–21, 2021.

[15] D. Pina, L. Kunstmann, D. de Oliveira, P. Valduriez, and M. Mattoso, "Provenance supporting hyperparameter analysis in deep neural networks," in *Provenance and Annotation of Data and Processes: 8th and 9th International Provenance and Annotation Workshop, IPAW 2020+ IPAW 2021, Virtual Event, July 19–22, 2021, Proceedings 8*. Springer, 2021, pp. 20–38.

[16] M. Herschel, R. Diestelkämper, and H. Ben Lahmar, "A survey on provenance: What for? what form? what from?" *The VLDB Journal*, vol. 26, pp. 881–906, 2017.

[17] R. Souza, L. Azevedo, R. Thiago, E. Soares, M. Nery, M. A. Netto, E. Vital, R. Cerqueira, P. Valduriez, and M. Mattoso, "Efficient runtime capture of multiworkflow data using provenance," in *2019 15th International Conference on eScience (eScience)*. IEEE, 2019, pp. 359–368.

[18] V. Silva, V. Campos, T. Guedes, J. Camata, D. de Oliveira, A. L. Coutinho, P. Valduriez, and M. Mattoso, "Dfanalyzer: runtime dataflow analysis tool for computational science and engineering applications," *SoftwareX*, vol. 12, p. 100592, 2020.

[19] M. Balazinska, S. Chaudhuri, A. Ailamaki, J. Freire, S. Krishnamurthy, and M. Stonebraker, "The next 5 years: what opportunities should the database community seize to maximize its impact?" in *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*, 2020, pp. 411–414.

[20] R. F. da Silva, R. M. Badia, V. Bala, D. Bard, P.-T. Bremer, I. Buckley, S. Caino-Lores, K. Chard, C. Goble, S. Jha *et al.*, "Workflows community summit 2022: A roadmap revolution," *arXiv preprint arXiv:2304.00019*, 2023.

[21] R. Han, S. Byna, H. Tang, B. Dong, and M. Zheng, "Prov-io: An i/o-centric provenance framework for scientific data on hpc systems," in *Proceedings of the 31st International Symposium on High-Performance Parallel and Distributed Computing*, 2022, pp. 213–226.

[22] Y. Tas, M. J. Baeth, and M. S. Aktas, "An approach to standalone provenance systems for big social provenance data," in *2016 12th International Conference on Semantics, Knowledge and Grids (SKG)*. IEEE, 2016, pp. 9–16.

[23] C. Adjih, E. Baccelli, E. Fleury, G. Harter, N. Mitton, T. Noel, R. Pissard-Gibollet, F. Saint-Marcel, G. Schreiner, J. Vandaele *et al.*, "Fit iot-lab: A large scale open experimental iot testbed," in *2015 IEEE 2nd World Forum on Internet of Things (WF-IoT)*. IEEE, 2015, pp. 459–464.

[24] R. Bolze, F. Cappello, E. Caron, M. Dayde, F. Desprez, E. Jeannot, Y. Jégou, S. Lanteri, J. Leduc, N. Melab, G. Mornet, R. Namyst, P. Primet, B. Quétier, O. Richard, E.-G. Talbi, and I. Touche, "Grid'5000: A Large Scale And Highly Reconfigurable Experimental Grid Testbed," *International Journal of High Performance Computing Applications*, vol. 20, no. 4, pp. 481–494, 2006. [Online]. Available: https://hal.inria.fr/hal-00684943

[25] N. Khoi Tran, B. Sabir, M. A. Babar, N. Cui, M. Abolhasan, and J. Lipman, "Proml: A decentralised platform for provenance management of machine learning software systems," in *Software Architecture: 16th European Conference, ECSA 2022, Prague, Czech Republic, September 19–23, 2022, Proceedings*. Springer, 2022, pp. 49–65.

[26] A. Spinuso, M. Atkinson, and F. Magnoni, "Active provenance for data-intensive workflows: engaging users and developers," in *2019 15th International Conference on eScience (eScience)*. IEEE, 2019, pp. 560–569.

[27] K. Xu, A. Ottley, C. Walchshofer, M. Streit, R. Chang, and J. Wenskovitch, "Survey on the analysis of user interactions and visualization provenance," in *Computer Graphics Forum*, vol. 39, no. 3. Wiley Online Library, 2020, pp. 757–783.

[28] K. Belhajjame, R. B'Far, J. Cheney, S. Coppens, S. Cresswell, Y. Gil, P. Groth, G. Klyne, T. Lebo, J. McCusker *et al.*, "Prov-dm: The prov data model," *W3C Recommendation*, vol. 14, pp. 15–16, 2013.

[29] P. Missier, K. Belhajjame, and J. Cheney, "The w3c prov family of specifications for modelling provenance metadata," in *Proceedings of the 16th International Conference on Extending Database Technology*, 2013, pp. 773–776.

[30] N. Garg, *Apache kafka*. Packt Publishing Birmingham, UK, 2013.

[31] M. Zaharia, M. Chowdhury, M. J. Franklin, S. Shenker, and I. Stoica, "Spark: Cluster Computing with Working Sets," in *Proceedings of the 2Nd USENIX Conference on Hot Topics in Cloud Computing*, ser. HotCloud'10. Berkeley, CA, USA: USENIX Association, 2010, pp. 10–10.

[32] B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. y Arcas, "Communication-efficient learning of deep networks from decentralized data," in *Artificial intelligence and statistics*. PMLR, 2017, pp. 1273–1282.

[33] Y. Ye, S. Li, F. Liu, Y. Tang, and W. Hu, "Edgefed: Optimized federated learning based on edge computing," *IEEE Access*, vol. 8, pp. 209 191–209 198, 2020.

[34] D. Pina, L. Kunstmann, D. de Oliveira, P. Valduriez, and M. Mattoso, "Provenance supporting hyperparameter analysis in deep neural networks," in *Provenance and Annotation of Data and Processes*. Springer, 2020, pp. 20–38.

[35] D. Rosendo, P. Silva, M. Simonin, A. Costan, and G. Antoniu, "E2Clab: Exploring the Computing Continuum through Repeatable, Replicable and Reproducible Edge-to-Cloud Experiments," in *Cluster 2020 - IEEE International Conference on Cluster Computing*, Kobe, Japan, Sep. 2020, pp. 1–11. [Online]. Available: https://hal.archives-ouvertes.fr/hal-02916032

[36] R.-A. Cherrueau, M. Delavergne, A. van Kempen, A. Lebre, D. Pertin, J. Rojas Balderrama, A. Simonet, and M. Simonin, "EnosLib: A Library for Experiment-Driven Research in Distributed Computing," *IEEE Transactions on Parallel and Distributed Systems*, vol. 33, no. 6, pp. 1464–1477, Jun. 2022. [Online]. Available: https://hal.inria.fr/hal-03324177

[37] K. Keahey, J. Anderson, Z. Zhen, P. Riteau, P. Ruth, D. Stanzione, M. Cevik, J. Colleran, H. S. Gunawi, C. Hammock *et al.*, "Lessons learned from the chameleon testbed," in *2020 USENIX Annual Technical Conference (USENIX ATC 20)*, 2020, pp. 219–233.

[38] P. Ruan, T. T. A. Dinh, Q. Lin, M. Zhang, G. Chen, and B. C. Ooi, "Lineagechain: a fine-grained, secure and efficient data provenance system for blockchains," *The VLDB Journal*, vol. 30, no. 1, pp. 3–24, 2021.

[39] T. Pasquier, X. Han, T. Moyer, A. Bates, O. Hermant, D. Eyers, J. Bacon, and M. Seltzer, "Runtime analysis of whole-system provenance," in *Proceedings of the 2018 ACM SIGSAC conference on computer and communications security*, 2018, pp. 1601–1616.

[40] F. F. de Mendonça Junior, Z. Kokkinogenis, K. L. Dias, P. M. d'Orey, and R. J. Rossetti, "The trade-offs between fog processing and communications in latency-sensitive vehicular fog computing," *Pervasive and Mobile Computing*, vol. 84, p. 101638, 2022.

[41] S. A. A. Abir, A. Anwar, J. Choi, and A. Kayes, "Iot-enabled smart energy grid: Applications and challenges," *IEEE access*, vol. 9, pp. 50 961–50 981, 2021.

[42] W.-C. Lo, C.-Y. Huang, and C.-H. Hsu, "Edge-assisted rendering of 360 videos streamed to head-mounted virtual reality," in *2018 IEEE International Symposium on Multimedia (ISM)*. IEEE, 2018, pp. 44–51.

[43] T. Zhang, C. He, T. Ma, L. Gao, M. Ma, and S. Avestimehr, "Federated learning for internet of things," in *Proceedings of the 19th ACM*

Conference on Embedded Networked Sensor Systems, 2021, pp. 413–419.

[44] Y. Liu, S. Garg, J. Nie, Y. Zhang, Z. Xiong, J. Kang, and M. S. Hossain, "Deep anomaly detection for time-series data in industrial iot: A communication-efficient on-device federated learning approach," *IEEE Internet of Things Journal*, vol. 8, no. 8, pp. 6348–6358, 2020.

[45] R. Ito, M. Tsukada, and H. Matsutani, "An on-device federated learning approach for cooperative model update between edge devices," *IEEE Access*, vol. 9, pp. 92 986–92 998, 2021.

[46] IoT-LAB, "Iot-lab a8-m3 board." https://www.iot-lab.info/docs/boards/iot-lab-a8-m3/, January 2, 2023.

[47] (2023, jan) Provenance capture iot/edge: experiment artifacts. [Online]. Available: https://gitlab.inria.fr/E2Clab/examples/provenance-iot-edge

[48] G5K, "Grid'5000: a large-scale and flexible testbed for experiment-driven research." https://www.grid5000.fr/w/Nancy:Hardware#gros, January 2, 2023.

[49] (2023, jul) Mqtt: The standard for iot messaging. [Online]. Available: https://mqtt.org/

[50] (2023, jul) Coap: The constrained application protocol. [Online]. Available: https://datatracker.ietf.org/doc/html/rfc7252

[51] (2023, jul) Amqp: Advanced message queuing protocol. [Online]. Available: http://docs.oasis-open.org/amqp/core/v1.0/os/amqp-core-overview-v1.0-os.html

[52] (2023, jul) Udp: User datagram protocol. [Online]. Available: https://datatracker.ietf.org/doc/html/rfc768

[53] (2023, jul) Rpl: Ipv6 routing protocol for low-power and lossy networks. [Online]. Available: https://datatracker.ietf.org/doc/html/rfc6550

[54] R. Morabito, Z. Laaroussi, and J. Jiménez, "Evaluating the performance of coap, mqtt, and http in vehicular scenarios," in *IEEE Conference on Standards for Communications and Networking, CSCN*, 2018.

[55] B. Wukkadada, K. Wankhede, R. Nambiar, and A. Nair, "Comparison with http and mqtt in internet of things (iot)," in *2018 International Conference on Inventive Research in Computing Applications (ICIRCA)*. IEEE, 2018, pp. 249–253.

[56] C. B. Gemirter, Ç. Şenturca, and Ş. Baydere, "A comparative evaluation of amqp, mqtt and http protocols using real-time public smart city data," in *2021 6th International Conference on Computer Science and Engineering (UBMK)*. IEEE, 2021, pp. 542–547.

[57] A. Stanford-Clark and H. L. Truong, "Mqtt for sensor networks (mqtt-sn) protocol specification," *International business machines (IBM) Corporation version*, vol. 1, no. 2, pp. 1–28, 2013.

[58] (2013, jan) Rsmb: Really small message broker. [Online]. Available: https://github.com/eclipse/mosquitto.rsmb

[59] (2017, jan) Eclipse mosquitto: An open source mqtt broker. [Online]. Available: https://github.com/eclipse/mosquitto

[60] (2017, jan) Python client for mqtt-sn brokers. [Online]. Available: https://github.com/luanguimaraesla/mqttsn

[61] (2018, jan) Requests: Http for humans. [Online]. Available: https://github.com/psf/requests

[62] (2018, jan) Dfanalyzer tool. [Online]. Available: https://gitlab.com/ssvitor/dataflow_analyzer

[63] Docker, "What is in a docker container?" https://www.docker.com/, January 2, 2023.

[64] P. A. Boncz, M. Zukowski, and N. Nes, "Monetdb/x100: Hyper-pipelining query execution." in *Cidr*, vol. 5, 2005, pp. 225–237.

[65] E. Deelman, K. Vahi, G. Juve, M. Rynge, S. Callaghan, P. J. Maechling, R. Mayani, W. Chen, R. F. Da Silva, M. Livny *et al.*, "Pegasus, a workflow management system for science automation," *Future Generation Computer Systems*, vol. 46, pp. 17–35, 2015.

[66] B. Ludäscher, I. Altintas, C. Berkley, D. Higgins, E. Jaeger, M. Jones, E. A. Lee, J. Tao, and Y. Zhao, "Scientific workflow management and the kepler system," *Concurrency and computation: Practice and experience*, vol. 18, no. 10, pp. 1039–1065, 2006.

[67] T. Oinn, M. Greenwood, M. Addis, M. N. Alpdemir, J. Ferris, K. Glover, C. Goble, A. Goderis, D. Hull, D. Marvin *et al.*, "Taverna: lessons in creating a workflow environment for the life sciences," *Concurrency and computation: Practice and experience*, vol. 18, no. 10, pp. 1067–1100, 2006.

[68] E. Deelman, S. Callaghan, E. Field, H. Francoeur, R. Graves, N. Gupta, V. Gupta, T. H. Jordan, C. Kesselman, P. Maechling *et al.*, "Managing large-scale workflow execution from resource provisioning to provenance tracking: The cybershake example," in *2006 Second IEEE International Conference on e-Science and Grid Computing (e-Science'06)*. IEEE, 2006, pp. 14–14.

[69] E. Nwafor, A. Campbell, D. Hill, and G. Bloom, "Towards a provenance collection framework for internet of things devices," in *2017 IEEE SmartWorld, Ubiquitous Intelligence & Computing, Advanced & Trusted Computed, Scalable Computing & Communications, Cloud & Big Data Computing, Internet of People and Smart City Innovation (SmartWorld/SCALCOM/UIC/ATC/CBDCom/IOP/SCI)*. IEEE, 2017, pp. 1–6.

[70] N. C. Narendra, A. Shukla, S. Nayak, A. Jagadish, and R. Kalkur, "Genoma: Distributed provenance as a service for iot-based systems," in *2019 IEEE 5th World Forum on Internet of Things (WF-IoT)*. IEEE, 2019, pp. 755–760.