Methodologies for Reproducible Analysis of Workflows on the Edge-to-Cloud Continuum

Daniel Rosendo

COLLEGE MATHS, TELECOMS
DOCTORAL INFORMATIQUE, SIGNAL
BRETAGNE SYSTEMES, ELECTRONIQUE

INSA INSTITUT NATIONAL DES SCIENCES APPLIQUÉES RENNES

THÈSE DE DOCTORAT DE

L'INSTITUT NATIONAL DES SCIENCES APPLIQUÉES RENNES

ÉCOLE DOCTORALE N° 601
Mathématiques, télécommunications, informatique, signal, systémes, électronique
Spécialité : *Informatique*

Par
Daniel ROSENDO

Methodologies for Reproducible Analysis of Workflows on the Edge-to-Cloud Continuum

Thèse présentée et soutenue à Rennes, le 01 Juin 2023
Unité de recherche : Inria
Thèse N° : 23ISAR 13 / D23 - 13

Rapporteurs avant soutenance :

Rosa BADIA	Research Director, Barcelona Supercomputing Center (BSC) - Spain
Ewa DEELMAN	Research Director, USC Information Sciences Institute - California, United States

Composition du Jury :

Président :	Maria PEREZ	Full Professor, Universidad Politécnica de Madrid (UPM) - Spain
Examinateurs :	Rosa BADIA	Research Director, Barcelona Supercomputing Center (BSC) - Spain
	Ewa DEELMAN	Research Director USC Information Sciences Institute - California, United States
Dir. de thèse :	Gabriel ANTONIU	Directeur de recherche, Inria - Rennes, France
Co-dir. de thèse :	Alexandru COSTAN	Maître de conférences HDR, INSA - Rennes, France
Co-dir. de thèse :	Patrick VALDURIEZ	Directeur de recherche, Inria - Montpellier, France

To my grandmothers.

To my family,
who make it all worthwhile.

Those who walk alone may arrive faster,
but those who go with others will certainly go further.
- Clarice Lispector

ACKNOWLEDGEMENT

I want to start by thanking my reviewers: Rosa Badia and Ewa Deelman, as well as the other members of my jury: Maria Perez, for taking the time to evaluate my Ph.D. thesis. Thanks also to the members of my CSID: Alexandre Termier and Bruno Raffin, for their valuable feedback.

This work is possible thanks to the continuous support and advice from my brilliant advisors, Gabriel Antoniu, Alexandru Costan, and Patrick Valduriez, who shared their energy and connections with great researchers and gave me the freedom to follow my research ideas. Thank you for every word that gave me the strength to go forward.

I owe the work presented here to the various great collaborations:

Thanks to Pedro Silva (from the Hasso-Plattner-Institut, Germany) for sharing his ideas and Matthieu Simonin (Inria) for the technical support and impressive response time. The Pl@ntNet team (University of Montpellier, LIRMM), in the context of the HPC-Big Data Project, in unique Alexis Joly and Jean-Christophe, for providing the use case to validate this work.

The Federal University of Rio de Janeiro (HPDeSc associate team with Brazil), and the Oak Ridge National Laboratory, particularly Marta Mattoso, Debora Pina, and Renan Souza, for the fruitful discussions and technical support.

I am also very grateful to Gabriel and Kate Keahey for hosting me (summer internship) at Argonne National Laboratory (ANL) and for the fruitful discussions with Kate, Michael, Mark, Adam, and Zhuo. The JLESC workshop allowed me to present and discuss my research results with researchers and interns at ANL, in particular Romain Egele, Jaehoon Koo, Prasanna Balaprakash, and Orcun Yildiz.

Thanks to the members of the KerData team: Luc Bougé (for his golden rules), Ovidiu, Luis, François, Josh, Thomas (for motivating me to start running), Julien, Cédric, Juliette, Hugo, Matthieu, José, and Edgar. Thanks to Gaelle and Laurence for helping me with paperwork and organizing my missions. Thanks to Esther, Reza, Benjamin, Joaquim, Alena, Cesar, and Camille from the Zenith team. Thanks to Luis, Johanne, and Maël from LACODAM team. I happily recall our quality time together during lunch, conferences, and events organized at Inria by AGOS.

Thanks to my Master advisors, Judith Kelner, Patricia Endo, and Djamel, for hosting me at the Networking and Telecommunications Research Group in Brazil and preparing me to start my research career and Ph.D. journey at Inria.

Finally, I would like to thank my beloved grandmothers (who taught me values I will carry with me for the rest of my life), my parents, my brothers, and all my family for their continuous encouragement, support, and help in every step that I make. You make it all worthwhile.

RÉSUMÉ EN FRANÇAIS

Contexte

La révolution numérique actuelle a un impact sur les êtres humains dans leur façon de vivre, de travailler, d'apprendre et de communiquer. Elle a entraîné des progrès impressionnants dans de nombreux domaines, tels que le Cloud Computing, le Calcul à Haute Performance (HPC), l'Intelligence Artificielle (IA), l'Analyse des Données Massives (Big Data) et l'Internet des Objets (IoT). En outre, de nouveaux scénarios d'applications stimulants apparaissent dans divers domaines tels que les véhicules autonomes, la fabrication en temps réel, l'agriculture de précision et les villes intelligentes, pour n'en citer que quelques-uns [223, 142].

L'explosion des données générées par ces applications et la nécessité d'analyse en temps réel et de prise de décision rapide ont entraîné un changement des paradigmes de traitement des données et de Machine Learning (ML) depuis des approches centralisées vers des infrastructures et services informatiques décentralisés [137].

Les workflows de traitement de données et de ML ne peuvent plus s'appuyer sur des approches traditionnelles [175] qui envoient toutes les données vers des centres de données Cloud centralisés et distants pour le traitement ou l'entraînement et l'inférence de modèles de ML. Au contraire, elles doivent exploiter les nombreuses ressources à proximité des sites de génération des données (*i.e.*, dans le Edge ou le Fog) pour extraire rapidement des informations [18] et satisfaire aux exigences de latence très faible des applications. Le défi est d'autant plus grand qu'il faut maintenir une utilisation raisonnable des ressources et préserver les contraintes de confidentialité. En pratique, pour équilibrer des exigences contradictoires, il est judicieux de peser les avantages respectifs de la centralisation et de la décentralisation et de faire les compromis appropriés pour utiliser intelligemment les avantages de chaque type d'infrastructure.

Cette approche flexible contribue à l'émergence du *continuum Edge-Cloud* [81], qui combine de manière transparente les ressources et les services du Cloud avec ceux à la périphérie du réseau dans le Edge et le Fog. Ainsi, les données sont d'abord générées et prétraitées (*e.g.*, filtrage, inférence de base) sur les dispositifs IoT/Edge, tandis que les nœuds de Fog traitent les données partiellement agrégées. Ensuite, si nécessaire, les données sont transférées dans le cloud pour des analyses Big Data, l'entraînement de modèles de ML et des simulations globales. Par exemple, les jumeaux numériques collectent les données dans le Edge et les stockent dans le Cloud, où ils exécutent l'analyse des données et l'apprentissage des modèles, et finalement exécutent des simulations basées sur ces modèles dans des clusters HPC.

En raison de la complexité des déploiements d'applications dans des infrastructures Edge-Cloud distribuées et hétérogènes, la vision du continuum Edge-Cloud reste à concrétiser.

Comprendre et optimiser les performances d'applicatons réelles à grande échelle dans ce contexte complexe est un défi majeur [175]. Cela nécessite de configurer une myriade de paramètres spécifiques au système (*e.g.*, systèmes d'IA et de Big Data, applications et systèmes d'ingestion, entre autres) et de concilier de nombreuses exigences ou contraintes en termes de consommation d'énergie, d'interopérabilité, de mobilité, de latence de communication, d'efficacité du réseau, de confidentialité des données et d'utilisation des ressources matérielles (*e.g.*, mémoire GPU, puissance CPU, taille de stockage, et autres) [231].

Cette thèse est une contribution conceptuelle et pratique au *continuum Edge-Cloud*, proposant des méthodologies et les appliquant dans des environnements novateurs. Nos méthodologies ont pour but de s'affranchir de la complexité de la compréhension et de l'optimisation des workflows dans le continuum Edge-Cloud. Ainsi, elles permettent la conception d'expériences reproductibles, l'optimisation des applications, la capture efficace des métadonnées de provenance des exécutions de workflows, et la reproductibilité des expérimentations. Nous avons validé notre proposition avec, d'abord, le développement du framework E2Clab qui supporte le cycle complet d'analyse d'une application dans le continuum Edge-Cloud, et ensuite, l'utilisation de E2Clab avec des applications réelles comme Pl@ntNet.

Contributions

Les principales contributions de cette thèse sont les suivantes:

Méthodologie pour la conception d'expériences reproductibles dans le continuum Edge-Cloud

Dans ce contexte, il est difficile de comprendre les performances de bout en bout. Cela revient à concilier de nombreuses exigences et contraintes applicatives, généralement contradictoires, avec des choix de conception d'infrastructure de bas niveau. Un défi crucial consiste à reproduire avec précision les comportements pertinents des workflows et les paramètres représentatifs de l'infrastructure physique sous-jascente. En d'autres termes, il faut trouver une approche rigoureuse pour répondre à des questions telles que : *Comment identifier les goulots d'étranglement de l'infrastructure ? Quels paramètres systèmes et quelles configurations d'infrastructures ont un impact sur les performances et comment ?*

Nous proposon une méthodologie rigoureuse pour un tel processus et la validons en développant la plateforme **E2C***lab*. Il s'agit de la première plateforme à prendre en charge le cycle complet d'analyse d'une application dans le continuum Edge-Cloud : (*i*) la configuration de l'environnement expérimental, des bibliothèques et des frameworks ; (*ii*) la correspondance

entre les parties de l'application et les machines dans le Edge, le Fog et le Cloud ; (*iii*) le déploiement de l'application sur l'infrastructure ; (*iv*) l'exécution automatisée ; et (*v*) la collecte des métriques de l'expérience. Nous illustrons son utilisation avec une application réelle déployée sur Grid'5000, montrant que notre framework permet de comprendre et d'améliorer les performances en les corrélant aux paramètres, à l'utilisation des ressources et aux spécificités de l'infrastructure sous-jacente.

Ce travail est une **collaboration** avec Pedro Silva du Hasso-Plattner-Institut (HPI), Université de Potsdam, Allemagne ; et Matthieu Simonin à Inria, France. L'article de référence **E2C***lab* a été **publié** à la conférence IEEE Cluster 2020, CORE Rank A [176]. **E2C***lab* est un logiciel open source, documenté, et a été utilisé et cité par la communauté des chercheurs [174].

Méthodologie d'optimisation des performances des applications Edge-Cloud

La nature automatisée des déploiements basés sur **E2C***lab* rend naturelle l'exploration des optimisations de performance des workflows Edge-Cloud. Ces workflows sont soumis à des contraintes et exigences complexes en matière de performances, d'utilisation des ressources, de consommation d'énergie et de coûts financiers. Il est donc difficile de répondre à des questions telles que : *Comment configurer le matériel et les composants du système pour minimiser la consommation d'énergie? Où les parties du workflows doivent-elles être exécutées dans le continuum Edge-Cloud pour minimiser les coûts de communication et la latence de bout en bout?*

Dans ce travail, nous proposons une méthodologie pour l'optimisation d'applications réelles dans le continuum Edge-Cloud. Nous l'avons implémenté dans le framework **E2C***lab*. Notre approche repose sur une analyse rigoureuse des configurations possibles dans un environnement de test contrôlé afin de comprendre leur comportement et les compromis de performance associés. Nous illustrons notre méthodologie en optimisant Pl@ntNet, une application mondiale d'identification des plantes. La validation expérimentale à grande échelle sur Grid'5000 montre que notre méthodologie s'est avérée utile pour comprendre et améliorer les performances de Pl@ntNet.

Ce travail est une **collaboration** avec Alexis Joly de l'équipe Pl@ntNet, Inria, Montpellier dans le cadre du Inria Project Lab (IPL) HPC-BigData et des discussions avec Orcun Yildiz et Romain Egele de l'Argonne National Laboratory - USA dans le cadre du Joint Laboratory for Extreme Scale Computing (JLESC). Cette contribution a donné lieu à une **publication** à la conférence IEEE Cluster 2021, CORE Rank A [181] et à un poster à IPDPS 2021, CORE Rank A [178].

Méthode de capture efficace des métadonnées de provenance pour les workflows s'éxécutant sur le continuum Edge-Cloud

La capture des métadonnées de provenance des indicateurs de performance clés, avec leurs données et processus associés, peut aider à comprendre et à optimiser les exécutions de workflows. Par exemple, elle peut aider à répondre à des questions telles que : *Après plusieurs évaluations de workflows, pouvons-nous comparer leur provenance et voir comment elle a changé?* ou *Quels paramètres de workflows ont produit ces résultats?*. Cependant, le coût de la capture des métadonnées de provenance peut être prohibitif, en particulier dans les appareils à ressources limitées, comme ceux du Edge.

Sur la base d'une analyse des performances des systèmes existants, nous proposons ProvLight, un outil permettant une capture efficace de la provenance dans le Edge. Nous tirons parti de modèles de données simplifiés, de la compression et du regroupement des données, et de protocoles de transmission légers pour réduire les coûts d'exécution. Nous avons intégré ProvLight dans E2Clab pour permettre la capture de la provenance du workflows dans le continuum Edge-Cloud. Cette intégration fait d'E2Clab une plateforme prometteuse pour l'optimisation des performances des applications par le biais d'expériences reproductibles. Nous validons ProvLight avec des charges de travail synthétiques sur des dispositifs IoT/Edge réels dans les testbeds à grande échelle Grid'5000 et FIT IoT LAB. Les évaluations montrent que ProvLight surpasse les meilleurs systèmes de provenance comme ProvLake et DfAnalyzer dans les dispositifs à ressources limitées. ProvLight est 26—37x plus rapide dans la capture et la transmission des données de provenance, utilise 5—7x moins de CPU, 2x moins de mémoire, transmet 2x moins de données et consomme 2—2,5x moins d'énergie.

Ce travail est une **collaboration** avec Marta Mattoso de l'Université Fédérale de Rio de Janeiro de l'équipe associée Inria HPDaSc avec le Brésil ; et Renan Souza du Oak Ridge National Laboratory - USA. Il est **publié** à la conférence IEEE Cluster 2023 [180], CORE Rank A.

Reproductibilité des expériences sur le continuum Edge-Cloud

Comprendre les compromis de performance des workflows à grande échelle déployés dans le continuum Edge-Cloud est un défi majeur. Pour y parvenir, il faut réaliser systématiquement des expériences pour permettre à d'autres chercheurs de reproduire l'étude et les conclusions obtenues sur différentes infrastructures. Cela se résume au processus fastidieux de conciliation des nombreuses exigences et contraintes expérimentales avec les choix de conception d'infrastructures de bas niveau. Cette contribution explore : *Comment permettre aux chercheurs de reproduire facilement des expériences complexes sur le continuum Edge-Cloud?* Ceci implique de permettre aux chercheurs de trouver et de partager facilement les artefacts d'expérience et de comprendre, reconfigurer et réaliser facilement les expériences.

Pour répondre aux limites des principales approches pour l'expérimentation distribuée et collaborative, telles que Google Colab, Kaggle et Code Ocean, nous proposons KheOps, un environnement collaboratif spécialement conçu pour permettre la reproductibilité des expériences sur le continuum Edge-Cloud. KheOps est composé de trois éléments centraux : (1) un répertoire d'expériences ; (2) un environnement de notebook ; et (3) une méthodologie d'expériences multiplateformes. Nous illustrons KheOps avec une application réelle. Les évaluations explorent le point de vue des auteurs d'une expérience décrite dans un article (qui visent à rendre leurs expériences reproductibles) et la perspective de leurs lecteurs (qui visent à reproduire l'expérience). Les résultats montrent comment KheOps aide les auteurs à réaliser systématiquement des expériences reproductibles sur Grid5000 + FIT IoT LAB. En outre, KheOps aide les lecteurs à reproduire facilement les expériences des auteurs dans différentes infrastructures telles que Chameleon Cloud + CHI@Edge, et à obtenir les mêmes conclusions avec une grande précision (>88% pour toutes les mesures de performance).

Ce travail est une **collaboration** avec Kate Keahey de l'Argonne National Laboratory - USA dans le cadre du Joint Laboratory for Extreme Scale Computing (JLESC). Cette contribution a donné lieu à une **publication** à la conférence ACM-REP'23 [179].

Publications

Revues Internationales

— **Daniel Rosendo**, Alexandru Costan, Patrick Valduriez, Gabriel Antoniu. Distributed intelligence on the continuum Edge-to-Cloud: A systematic literature review. **JPDC** - *Journal of Parallel and Distributed Computing*, *Elsevier*, 2022, 166, pp.71-94. **CORE Rank A**.

Conférences Internationales

— **Daniel Rosendo**, Kate Keahey, Alexandru Costan, Matthieu Simonin, Patrick Valduriez, Gabriel Antoniu. KheOps: Cost-effective Repeatability, Reproducibility, and Replicability of Edge-to-Cloud Experiments. **ACM REP 2023** - *ACM Conference on Reproducibility and Replicability*, Jun 2023, Santa Cruz California, United States.

— **Daniel Rosendo**, Marta Mattoso, Alexandru Costan, Renan Souza, Debora Pina, Patrick Valduriez, Gabriel Antoniu. ProvLight: Efficient Workflow Provenance Capture on the Edge-to-Cloud Continuum. **Cluster 2023** - *IEEE International Conference on Cluster Computing*, October 2023, Santa Fe, New Mexico, United States. **CORE Rank A** (acceptance rate 25%).

— **Daniel Rosendo**, Alexandru Costan, Gabriel Antoniu, Matthieu Simonin, Jean-Christophe Lombardo, Alexis Joly, Patrick Valduriez. Reproducible Performance Optimization of Com-

plex Applications on the Edge-to-Cloud Continuum. **Cluster 2021** - *IEEE International Conference on Cluster Computing*, Sep 2021, Portland, OR, United States. pp.23-34. **CORE Rank A** (acceptance rate 29%).

— **Daniel Rosendo**, Pedro Silva, Matthieu Simonin, Alexandru Costan, Gabriel Antoniu. E2Clab: Exploring the Computing Continuum through Repeatable, Replicable and Reproducible Edge-to-Cloud Experiments. **Cluster 2020** - *IEEE International Conference on Cluster Computing*, Sep 2020, Kobe, Japan. pp.1-11. **CORE Rank A** (acceptance rate 31%).

Posters dans des Conférences Internationales

— **Daniel Rosendo**, Alexandru Costan, Gabriel Antoniu, Patrick Valduriez. E2Clab: Reproducible Analysis of Complex Workflows on the continuum Edge-to-Cloud. **IPDPS 2021** - *35th IEEE International Parallel and Distributed Processing Symposium*, May 2021, Virtual, France. **CORE Rank A**.

Conférences Nationales

— **Daniel Rosendo**, Alexandru Costan, Gabriel Antoniu, Matthieu Simonin, Jean-Christophe Lombardo, Alexis Joly, Patrick Valduriez. Reproducible Performance Optimization of Complex Applications on the Edge-to-Cloud Continuum. **BDA 2022** - *38ème Conférence sur la Gestion de Données - Principes, Technologies et Applications*, Oct 2022, Clermont-Ferrand, France.

Posters dans des Conférences Nationales

— **Daniel Rosendo**, Alexandru Costan, Gabriel Antoniu, Patrick Valduriez. Enabling Reproducible Analysis of Complex Workflows on the Edge-to-Cloud Continuum. **BDA 2021** - *37ème Conférence sur la Gestion de Données - Principes, Technologies et Applications*, Oct 2021, Paris, France.

Logiciels

Contributions Principales

E2C*lab*

Description scientifique: E2C*Lab* est un framework qui permet aux chercheurs de reproduire de manière représentative le comportement des applications dans un environnement contrôlé pour des expériences approfondies et, par conséquent, de comprendre les performances de bout en bout des applications en corrélant les résultats aux paramètres. **E2C***lab* fournit une

approche rigoureuse pour répondre à des questions telles que : *Comment identifier les goulots d'étranglement de l'infrastructure ? Quels paramètres systèmes et quelles configurations d'infrastructures ont un impact sur les performances et comment?*

Description fonctionnelle: les caractéristiques de haut niveau fournies par **E2C***lab* sont : (*i*) Reproductibilité des expériences. (*ii*) Correspondance entre les parties applicatives (Edge, Fog et Cloud/HPC) et le testbed physique. (*iii*) Variation de l'expérience et mise à l'échelle transparente des scénarios. (*iv*) Émulation de réseau avec contraintes de communication Edge-to-Cloud. (*v*) Gestion des expériences : déploiement, exécution et surveillance (*e.g.,* sur Grid'5000, Chameleon et FIT IoT LAB). (*vi*) Optimisation : recherche de configuration des workflows des applications. (*vii*) Provenance : capture des données des workflows Edge-to-Cloud.

— **Link:** https://gitlab.inria.fr/E2Clab/e2clab

— **Taille et langage(s):** ∼3K lignes, Python.

— **Licence:** GNU General Public License v3.0

ProvLight

Description scientifique: ProvLight est un outil qui permet aux chercheurs de capturer efficacement les métadonnées de provenance des workflows qui s'exécutent dans des infrastructures IoT/Edge. ProvLight a une faible coût concernant le temps de capture, l'utilisation du processeur et de la mémoire, l'utilisation du réseau et la consommation d'énergie.

Description fonctionnelle: ProvLight adopte une architecture *master/worker* où le *master* reçoit les données capturées des *workers*, puis les traduit et les envoie aux systèmes de provenance. ProvLight fournit également une bibliothèque Python (conforme à la recommandation PROV-DM du W3C) qui permet aux utilisateurs de capturer les données de leurs workflows par l'instrumentation du code d'application.

— **Link:** https://gitlab.inria.fr/provlight/provlight

— **Taille et langage(s):** ∼700 lignes, Python.

— **Licence:** GNU General Public License v3.0

Contribution à un logiciel existant

EnOSlib

Description scientifique: EnOSlib est une bibliothèque Python pour la recherche expérimentale reproductible en informatique distribuée. Elle vise à aider les chercheurs dans le processus de développement de leurs artefacts expérimentaux et dans leur exécution et leur reproduction sur différentes infrastructures.

Description fonctionnelle: EnOSlib apporte des blocs de construction réutilisables pour la configuration de l'infrastructure, le provisionnement de logiciels sur les hôtes distants et l'organisation du workflow expérimental. L'interaction avec les testbeds (par exemple, Grid'5000, FIT IoT LAB et Chameleon) est confiée au fournisseur d'EnOSlib, et diverses actions sur les hôtes distants reposent également sur les mécanismes offerts par la bibliothèque.

Notre contribution: Nous permettons à EnOSlib de prendre en charge la gestion de workflows complexes Edge-to-Cloud sur Chameleon Cloud et CHI@Edge. Nous avons étendu l'abstraction des fournisseurs d'EnOSlib pour permettre aux chercheurs de louer des ressources (calcul, stockage et réseau) à partir des Chameleon testbed, ainsi que de déployer et d'exécuter leurs workflows.

— **Link:** https://gitlab.inria.fr/discovery/enoslib

— **Taille et langage(s):** ∼1K lignes, Python.

— **Licence:** GNU General Public License v3.0

Outils et artefacts orientés vers la reproductibilité

KheOps

Description scientifique: KheOps est un environnement collaboratif conçu pour permettre la reproductibilité des expériences Edge-to-Cloud dans les testbeds scientifiques à grande échelle. Il permet aux chercheurs de trouver et de partager facilement des artefacts d'expérience et de comprendre, reconfigurer et réaliser facilement des expériences.

Description fonctionnelle: KheOps est composé de trois éléments fondamentaux : (*i*) le portail de partage Trovi pour le conditionnement et le partage des artefacts ; (*ii*) les Jupyter notebooks pour combiner les processus d'expérience avec le code exécutable ; et (*iii*) la méthodologie **E2C***lab* pour réaliser des expériences dans les testbeds tels que Chameleon Cloud, CHI@Edge, Grid'5000 et FIT IoT LAB. L'objectif est de réduire la barrière à la reproduction de la recherche en combinant les artefacts et l'environnement expérimental et en fournissant un dépôt en libre accès des artefacts de recherche qui sont visibles et reproductibles à travers les testbeds.

— **Link:** https://gitlab.inria.fr/KheOps/kheops

— **Taille et langage(s):** ∼800 lignes, Python

— **Licence:** GNU General Public License v3.0

Organisation du Manuscrit

Ce manuscrit est organisé en trois parties principales et huit chapitres.

La première partie: Le chapitre 2 présente le contexte de notre recherche. Il introduit la pertinence de la reproductibilité dans la communauté informatique ainsi que les systèmes existants et les testbeds scientifiques à grande échelle pour la recherche expérimentale dans le continuum Edge-to-Cloud. Puis il discute des défis ouverts concernant l'optimisation reproductible des performances des workflows Edge-to-Cloud dans des infrastructures hétérogènes. Enfin, il souligne les opportunités qui ont motivé nos contributions.

La deuxième partie: Les chapitres 3 à 5 se concentrent sur les contributions liées à l'approche **E2C***lab*. Trois objectifs majeurs sont abordés : (*i*) des expériences reproductibles dans le continuum Edge-to-Cloud ; (*ii*) la compréhension des performances des workflows dans des infrastructures hétérogènes ; et (*iii*) l'optimisation des performances des workflows Edge-to-Cloud.

Le chapitre 3 présente nos méthodologies pour comprendre et optimiser les performances des workflows Edge-to-Cloud au moyen d'expériences reproductibles. L'implémentation de ces méthodologies dans le framework **E2C***lab* est présentée au chapitre 4. Les validations expérimentales à grande échelle de nos méthodologies avec des applications du monde réel (*e.g.*, application botanique Pl@ntNet) sont présentées au chapitre 5.

La troisième partie: Comme la compréhension, l'optimisation et la reproduction de workflows complexes s'exécutant sur le continuum Edge-Cloud peuvent être facilitées par la capture de métadonnées de provenance. Le chapitre 6 explore la capture efficace de données de workflows s'exécutant dans des infrastructures IoT/Edge aux ressources limitées. Il propose ProvLigh, une approche qui vise à capturer les métadonnées de provenance avec un faible surcoût. **E2C***lab* est étendu pour permettre la capture de données de provenance à l'exécution des workflows Edge-to-Cloud.

Le chapitre 7 propose KheOps, un environnement collaboratif conçu pour permettre la reproductibilité des expériences sur le continuum Edge-Cloud. KheOps comprend un répertoire d'expériences, un environnement de notebooks et une méthodologie d'expériences multiplateformes. KheOps est illustré avec une application réelle.

Le chapitre 8 conclut cette thèse et présente les nouvelles perspectives de recherche.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LISTINGS

Chapter 1

Introduction

Contents

1.1 Context

The current digital revolution is impacting human beings in the way they live, work, learn, and communicate. This has resulted in impressive progress in many areas, such as Cloud Computing, High-Performance Computing (HPC), Artificial Intelligence (AI), Big Data Analytics, and the Internet of Things (IoT). Furthermore, new challenging application scenarios are emerging from various domains such as autonomous vehicles, real-time manufacturing, precision agriculture, and smart cities, to cite just a few [223, 142].

The explosion of data generated by many applications in the aforementioned areas and the need for real-time analytics and fast decision-making has resulted in a shift of the data processing paradigms, as well as of Machine Learning (ML) paradigms, from centralized approaches towards decentralized and multi-tier computing infrastructures and services [137].

Data processing and AI workflows can no longer rely on traditional approaches [175] that send all data to centralized and distant Cloud datacenters for processing or AI model training and inference. Instead, they need to leverage the numerous resources close to the data generation sites (*i.e.*, in the Edge or Fog) to promptly extract insights [18] and satisfy the ultra-low latency requirements of applications. This is made more challenging by maintaining reasonable resource usage and preserving privacy constraints. In practice, to balance contradictory

requirements, in many situations, it makes sense to weigh the respective benefits of centralization and decentralization and make appropriate trade-offs to use the advantages of each type of infrastructure smartly.

This flexible approach contributes to the emergence of what is called the *Computing Continuum* [81] (or the *Digital 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.*, filtering, basic inference) on IoT/Edge devices, while Fog nodes further process partially aggregated data. Then, if required, data is transferred to HPC-enabled Clouds for Big Data analytics, Artificial Intelligence model training, and global simulations. For instance, Digital Twins collect data from the edge, store it in the cloud, where they run data analytics and model training, and eventually run simulations based on those models in HPC clusters.

Due to the complexity incurred by application deployments on such highly distributed and heterogeneous Edge-to-Cloud infrastructures, the Computing Continuum vision remains to be realized in practice. **Deploying**, **analyzing** and **reproducing** performance trade-offs, and **optimizing** large-scale, real-life applications on such infrastructures is challenging [175]. It requires configuring a myriad of system-specific parameters (*e.g.,* from AI and Big Data systems, applications, and ingestion systems, among others) and reconciling many requirements or constraints in terms of energy consumption, interoperability, mobility, communication latency, network efficiency, data privacy, and hardware resource usage (*e.g.,* GPU memory, CPU power, storage size, and others) [231].

In this thesis, we make a first step towards enabling the *Computing Continuum* vision by proposing methodologies that guide researchers to systematically deploy, understand and optimize performance and reproduce complex Edge-to-Cloud workflows on large-scale and heterogeneous infrastructures.

1.2 Contributions

The main contributions of this thesis can be summarized as follows:

A Methodology for Designing Reproducible Experiments on the Computing Continuum

In this complex continuum, understanding end-to-end performance is challenging. This breaks down to reconciling many, typically contradicting application requirements and constraints with low-level infrastructure design choices. One crucial challenge is to accurately reproduce relevant behaviors of a given application workflow and representative settings of the physical infrastructure underlying this complex continuum. That is, finding a rigorous approach

to answering questions like *How to identify infrastructure bottlenecks? Which system parameters and infrastructure configurations impact performance and how?*

We introduce a rigorous methodology for such a process and validate it through **E2C***lab*. It is the first platform to support the complete analysis cycle of an application on the Computing Continuum: (*i*) the configuration of the experimental environment, libraries, and frameworks; (*ii*) the mapping between the application parts and machines on the Edge, Fog and Cloud; (*iii*) the deployment of the application on the infrastructure; (*iv*) the automated execution; and (*v*) the gathering of experiment metrics. We illustrate its usage with a real-life application deployed on the Grid'5000 testbed, showing that our framework allows one to understand and improve performance by correlating it to the parameter settings, the resource usage, and the specifics of the underlying infrastructure.

This work is a **collaboration** with Pedro Silva from Hasso-Plattner-Institut (HPI), University of Potsdam, Germany; and Matthieu Simonin at Inria, France. The reference **E2C***lab* paper was **published** at the IEEE Cluster 2020 conference, CORE Rank A (please see [176]). **E2C***lab* is open source, documented, and has been used and cited by the research community (please see [174]).

A Methodology for Optimizing the Performance of Edge-to-Cloud Applications

The automated nature of the **E2C***lab* based deployments makes it seamless/natural to explore the performance optimizations of Edge-to-Cloud workflows. Such workflows are subject to complex constraints and requirements regarding performance, resource usage, energy consumption, and financial costs. This makes it challenging to answer questions like *How to configure the hardware and system components to minimize energy consumption? Where should the workflow parts be executed across the Edge-to-Cloud Continuum to minimize communication costs and end-to-end latency?*

In this work, we propose a methodology to support the optimization of real-life applications on the Edge-to-Cloud Continuum. We implement it atop **E2C***lab* framework. Our approach relies on a rigorous analysis of possible configurations in a controlled testbed environment to understand their behavior and related performance trade-offs. We illustrate our methodology by optimizing Pl@ntNet, a worldwide plant identification application. Large-scale experimental validation on the Grid'5000 testbed shows that our methodology has proved helpful for understanding and improving the performance of Pl@ntNet.

This work is a **collaboration** with Alexis Joly from the Pl@ntNet team at the University of Montpellier - France in the context of the HPC-BigData Inria Project Lab and discussions with Orcun Yildiz and Romain Egele at the Argonne National Laboratory - USA in the context of the Joint Laboratory for Extreme Scale Computing (JLESC). This contribution led to a **publication** at the IEEE Cluster 2021 conference, CORE Rank A (please see [181]) and a poster at IPDPS 2021, CORE Rank A (please see [178]).

Enabling the Efficient Workflow Provenance Capture on the Edge-to-Cloud Continuum

Capturing the provenance of key performance indicators, with their related data and processes, may assist in understanding and optimizing workflow executions. For instance, it may help answer questions like *After multiple workflow evaluations, can we compare their provenance and see how it has changed?* or *What workflow parameters produced these results?* However, the provenance capture overhead can be prohibitive, particularly in resource-constrained devices, such as the ones on the IoT/Edge.

Based on a performance analysis of the 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 with synthetic workloads on real-life IoT/Edge devices in the large-scale Grid'5000 and FIT IoT LAB testbeds. Evaluations show that ProvLight outperforms state-of-the-art provenance systems like ProvLake and DfAnalyzer in resource-constrained devices. ProvLight is 26—37x faster in capturing and transmitting provenance data, uses 5—7x less CPU, 2x less memory, transmits 2x less data, and consumes 2—2.5x less energy.

This work is a **collaboration** with Marta Mattoso from the Federal University of Rio de Janeiro and the HPDeSc associate team with Brazil; and Renan Souza from the Oak Ridge National Laboratory - USA. It is **published** at IEEE Cluster 2023 conference, CORE Rank A (please see [180]).

Cost-effective Repeatability, Reproducibility, and Replicability of Edge-to-Cloud Experiments

Understanding the performance trade-offs of large-scale workflows deployed on the Edge-to-Cloud Continuum is challenging. To achieve this, one needs to systematically perform experiments to enable their reproducibility and allow other researchers to replicate the study and the obtained conclusions on different infrastructures. This breaks down to the tedious process of reconciling the numerous experimental requirements and constraints with low-level infrastructure design choices. This contribution explores: *How to allow researchers to reproduce and replicate complex Edge-to-Cloud experiments cost-effectively?* Cost-effective means to allow researchers to easily find and share experiment artifacts and easily understand, reconfigure, and perform the experiments.

To address the limitations of the main state-of-the-art approaches for distributed, collaborative experimentation, such as Google Colab, Kaggle, and Code Ocean, we propose KheOps, a collaborative environment specifically designed to enable cost-effective reproducibility and

replicability of Edge-to-Cloud experiments. KheOps is composed of three core elements: (1) an experiment repository; (2) a notebook environment; and (3) a multi-platform experiment methodology. We illustrate KheOps with a real-life Edge-to-Cloud application. The evaluations explore the point of view of the authors of an experiment described in an article (who aim to make their experiments reproducible) and the perspective of their readers (who aim to replicate the experiment). The results show how KheOps helps authors to systematically perform repeatable and reproducible experiments on the Grid5000 + FIT IoT LAB testbeds. Furthermore, KheOps helps readers to cost-effectively replicate authors experiments in different infrastructures such as Chameleon Cloud + CHI@Edge testbeds, and obtain the same conclusions with high accuracies (>88% for all performance metrics).

This work is a **collaboration** with Kate Keahey from the Argonne National Laboratory - USA in the context of the Joint Laboratory for Extreme Scale Computing (JLESC). This contribution led to a **publication** at ACM-REP'23 conference (please see [179]).

1.3 Publications

Journal Articles

— **Daniel Rosendo**, Alexandru Costan, Patrick Valduriez, Gabriel Antoniu. Distributed intelligence on the Edge-to-Cloud Continuum: A systematic literature review. **JPDC** - *Journal of Parallel and Distributed Computing*, *Elsevier*, 2022, 166, pp.71-94. **CORE Rank A**.

International Conferences

— **Daniel Rosendo**, Kate Keahey, Alexandru Costan, Matthieu Simonin, Patrick Valduriez, Gabriel Antoniu. KheOps: Cost-effective Repeatability, Reproducibility, and Replicability of Edge-to-Cloud Experiments. **ACM REP 2023** - *ACM Conference on Reproducibility and Replicability*, Jun 2023, Santa Cruz California, United States.

— **Daniel Rosendo**, Marta Mattoso, Alexandru Costan, Renan Souza, Debora Pina, Patrick Valduriez, Gabriel Antoniu. ProvLight: Efficient Workflow Provenance Capture on the Edge-to-Cloud Continuum. **Cluster 2023** - *IEEE International Conference on Cluster Computing*, October 2023, Santa Fe, New Mexico, United States. **CORE Rank A** (acceptance rate 25%).

— **Daniel Rosendo**, Alexandru Costan, Gabriel Antoniu, Matthieu Simonin, Jean-Christophe Lombardo, Alexis Joly, Patrick Valduriez. Reproducible Performance Optimization of Complex Applications on the Edge-to-Cloud Continuum. **Cluster 2021** - *IEEE International Conference on Cluster Computing*, Sep 2021, Portland, OR, United States. pp.23-34. **CORE Rank A** (acceptance rate 29%).

— **Daniel Rosendo**, Pedro Silva, Matthieu Simonin, Alexandru Costan, Gabriel Antoniu. E2Clab: Exploring the Computing Continuum through Repeatable, Replicable and Reproducible Edge-to-Cloud Experiments. **Cluster 2020** - *IEEE International Conference on Cluster Computing*, Sep 2020, Kobe, Japan. pp.1-11. **CORE Rank A** (acceptance rate 31%).

Posters at International Conferences

— **Daniel Rosendo**, Alexandru Costan, Gabriel Antoniu, Patrick Valduriez. E2Clab: Reproducible Analysis of Complex Workflows on the Edge-to-Cloud Continuum. **IPDPS 2021** - *35th IEEE International Parallel and Distributed Processing Symposium*, May 2021, Virtual, France. **CORE Rank A**.

National Conferences

— **Daniel Rosendo**, Alexandru Costan, Gabriel Antoniu, Matthieu Simonin, Jean-Christophe Lombardo, Alexis Joly, Patrick Valduriez. Reproducible Performance Optimization of Complex Applications on the Edge-to-Cloud Continuum. **BDA 2022** - *38ème Conférence sur la Gestion de Données - Principes, Technologies et Applications*, Oct 2022, Clermont-Ferrand, France.

Posters at National Conferences

— **Daniel Rosendo**, Alexandru Costan, Gabriel Antoniu, Patrick Valduriez. Enabling Reproducible Analysis of Complex Workflows on the Edge-to-Cloud Continuum. **BDA 2021** - *37ème Conférence sur la Gestion de Données - Principes, Technologies et Applications*, Oct 2021, Paris, France.

1.4 Software

1.4.1 Main Contributions

E2C*lab*

Scientific Description: E2C*lab* is a framework that allows researchers to reproduce in a representative way the application behavior in a controlled environment for extensive experiments and, therefore, to understand the end-to-end performance of applications by correlating results to the parameter settings. **E2C*lab*** provides a rigorous approach to answering questions like: *How to identify infrastructure bottlenecks? Which system parameters and infrastructure configurations impact performance and how?*

Functional Description: High-level features provided by **E2C***lab*: (*i*) Reproducible Experiments: Supports repeatability, replicability and reproducibility. (*ii*) Mapping: Application parts (Edge, Fog, and Cloud/HPC) and physical testbed. (*iii*) Variation & Scaling: Experiment variation and transparent scaling of scenarios. (*iv*) Network Emulation: Edge-to-Cloud communication constraints. (*v*) Experiment Management: Deployment, execution and monitoring (*e.g.*, on Grid'5000, Chameleon, and FIT IoT LAB). (*vi*) Optimization: configuration search of application workflows. (*vii*) Provenance: data capture of Edge-to-Cloud workflows.

— **Link:** https://gitlab.inria.fr/E2Clab/e2clab

— **Size and language(s):** ∼3K lines, Python.

— **License:** GNU General Public License v3.0

ProvLight

Scientific Description: ProvLight is a framework that allows researchers to efficiently capture provenance data of workflows running on IoT/Edge infrastructures. ProvLight presents low capture overhead regarding capture time, CPU and memory usage, network usage, and power consumption.

Functional Description: ProvLight follows a *master/worker* architecture where the *master* receives the captured data from *workers* and then translates and sends to provenance systems. ProvLight also provides a Python library (which follows the W3C PROV-DM recommendation) that allows users to capture data from their workflows (through application code instrumentation).

— **Link:** https://gitlab.inria.fr/provlight/provlight

— **Size and language(s):** ∼700 lines, Python.

— **License:** GNU General Public License v3.0

1.4.2 Contribution to Existing Software

EnOSlib

Scientific Description: EnOSlib is a Python library focusing on reproducible-driven experimental research in distributed computing. It aims at helping researchers in the process of developing their experimental artifacts and running and reproducing them over different infrastructures.

Functional Description: EnOSlib brings reusable building blocks for configuring the infrastructure, provisioning software on remote hosts, and organizing the experimental workflow. Interaction with the testbeds (*e.g.*, Grid'5000, FIT IoT LAB, and Chameleon) is deferred to

EnOSlib's provider, and various actions on remote hosts also rely on mechanisms offered by the library.

Our Contribution: We enable EnOSlib to support the management of complex Edge-to-Cloud workflows on the Chameleon Cloud and CHI@Edge testbeds. We extended EnOSlib's *Providers* abstraction to enable researchers to lease resources (*e.g.*, compute, storage, and networking) from the Chameleon testbeds, as well as deploy and execute their workflows.

— **Link:** https://gitlab.inria.fr/discovery/enoslib

— **Size and language(s):** ∼1K lines, Python.

— **License:** GNU General Public License v3.0

1.4.3 Reproducibility-oriented Tools and Artifacts

KheOps

Scientific Description: KheOps is a collaborative environment designed to enable cost-effective reproducibility and replicability of Edge-to-Cloud experiments in large-scale scientific testbeds. It allows researchers to easily find and share experiment artifacts and easily understand, reconfigure, and perform experiments.

Functional Description: KheOps is composed of three core elements: (*i*) Trovi sharing portal for packaging and sharing artifacts; (*ii*) Jupyter notebooks for combining experiment processes with executable code; and (*iii*) **E2C***lab* methodology for performing experiments in testbeds such as Chameleon Cloud, CHI@Edge, Grid'5000, and FIT IoT LAB testbeds. The goal is to lower the barrier to reproducing research by combining the artifacts and the experimental environment and providing an open-access repository of research artifacts that are visible and reproducible across testbeds.

— **Link:** https://gitlab.inria.fr/KheOps/kheops

— **Size and language(s):** ∼800 lines, Python

— **License:** GNU General Public License v3.0

1.5 Organization of the Manuscript

This manuscript is organized into three main parts and eight chapters.

The first part: Chapter 2 presents the context of our research. It introduces the relevance of Reproducibility in the Computer Science community and the existing systems and large-scale scientific testbeds for experimental research on the Edge-to-Cloud Continuum. Then, it discusses the open challenges regarding reproducible performance optimization of Edge-to-Cloud

workflows on highly heterogeneous infrastructures. Finally, it highlights the opportunities that drove our contributions.

The second part: Chapters 3 to 5 focus on contributions related to the **E2C***lab* approach. Three major objectives are being tackled: (*i*) reproducible experiments on the Computing Continuum; (*ii*) understanding performance of workflows on heterogeneous infrastructures; and (*iii*) performance optimization of Edge-to-Cloud workflows.

Chapter 3 introduces our methodologies for understanding and optimizing the performance of Edge-to-Cloud workflows through reproducible experiments. Implementation of these methodologies in the **E2C***lab* framework is presented in Chapter 4. Large-scale experimental validations of our methodologies with real-world applications (*e.g.,* Pl@ntNet botanical application) are presented in Chapter 5.

The third part: As understanding, optimizing, and reproducing complex Edge-to-Cloud workflows may be assisted by provenance data capture, Chapter 6 explores the efficient data capture of workflows running on resource-constrained IoT/Edge infrastructures. It proposes ProvLigh, an approach that aims to capture provenance with low overhead. **E2C***lab* is extended to enable runtime provenance data capture of Edge-to-Cloud workflows.

Chapter 7 proposes KheOps, a collaborative environment designed to enable cost-effective reproducibility and replicability of Edge-to-Cloud experiments. KheOps comprises an experiment repository, a notebook environment, and a multi-platform experiment methodology. It illustrates KheOps with a real-life Edge-to-Cloud application.

Chapter 8 concludes this thesis and presents the prospects brought by our solutions.

Part I

Context: Reproducibility and Experimental Research on the Edge-to-Cloud Continuum

Chapter 2

Background

Contents

This thesis focuses on reproducibility and experimental research on the Edge-to-Cloud Continuum. As introduced earlier, the Computing Continuum is a new computing paradigm that brings new research challenges and opportunities in many domains of interest. Therefore, we start by analyzing the state-of-the-art in these domains.

This chapter provides an overview of the significant findings of our systematic study [175] of state-of-the-art systems and open scientific testbeds for reproducible experimental research on the Computing Continuum. First, it defines the main terms used in this work and then presents how the selected articles in this review support the reproducibility of the experiments. In addition, it discusses the relevant performance metrics typically considered in Edge-to-Cloud experiments and introduces the definition of optimization problems. Next, it summarizes the main systems and large-scale scientific testbeds for Edge-to-Cloud experimental research. Finally, it discusses relevant open research challenges explored in this thesis.

Table 2.1 – ACM Digital Library Terminology Version 1.1 [17]

Repeatability	*Same team, same experimental setup: the measurement can be obtained with stated precision by the same team using the same measurement procedure, the same measuring system, under the same operating conditions, in the same location on multiple trials. For computational experiments, this means that a researcher can reliably repeat their own computation.*
Reproducibility	*Different team, same experimental setup: the measurement can be obtained with stated precision by a different team using the same measurement procedure, the same measuring system, under the same operating conditions, in the same or a different location on multiple trials. For computational experiments, this means that an independent group can obtain the same result using the author's own artifacts.*
Replicability	*Different team, different experimental setup: the measurement can be obtained with stated precision by a different team, a different measuring system, in a different location on multiple trials. For computational experiments, this means that an independent group can obtain the same result using artifacts which they develop completely independently.*

2.1 Reproducibility in Computing Continuum Research

2.1.1 Definitions and Landscape

As communities from an increasing number of scientific domains are leveraging the Computing Continuum, a desired feature of any experimental research is that its scientific claims are verifiable by others to build upon them. This can be achieved through repeatability, reproducibility, and replicability (3R's) [27, 212]. There are many non-uniform definitions of the 3Rs in literature. In this work, we follow the terminology proposed by the ACM Digital Library [17] (Artifact Review and Badging), as presented in Table 2.1.

Achieving **repeatability** means that one can reliably repeat the experiments and obtain precise measurements (*e.g.*, Edge to Cloud processing latency, memory consumption, among others) by using the same methodology and artifacts (*i.e.*, same testbed, same physical machines, same libraries/framework, same network configuration). Executing multiple experiments allows us to explore different scenario settings (*e.g.*, varying the number of Edge devices) and explore the impact of various parameters (*e.g.*, the network configuration between Edge devices and the Cloud server) on the performance metrics.

Reproducibility means that external researchers having access to the original methodology

(*e.g.*, configuration of physical machines, network, and systems, scenario descriptions) and using their artifacts (*i.e.*, data sets, scripts, AI frameworks, *etc.*) can obtain precise measurements of the application processing latency and throughput, for instance.

Replicability refers to independent researchers (*i.e.*, the readers of an article that was published by a different team) having access to the original methodology and artifacts (*e.g.*, configuration of physical machines, processing steps, network setup, *etc.*) and performing the experiments in different testbeds. The goal is that independent researchers can obtain precise results and conclusions consistent with the original study.

The selected studies. This systematic study follows a review methodology based on [122, 80]. First, we define the research questions focusing on the Edge-to-Cloud Continuum. Next, we define the search string to find articles on scientific databases such as ACM, IEEE Xplore, Springer, among others. Then, we define the inclusion/exclusion criteria and start the screening process, which consists in reading the abstract and conclusions for each article. Finally, **we select a total of 69 studies**.

Most of the challenges of achieving research 3Rs may be divided into three main categories: the need for a well-defined experimentation methodology, access to experiment artifacts, and access to experiment results.

How do the selected studies support the reproducibility of the experiments? We evaluate the support for the reproducibility of experiments for each selected article. This evaluation is based on the following three main relevant aspects:

— **Access to artifacs**: if authors provide access to a public repository with the artifacts used to run the experiments, such as datasets, codes, applications, systems, configuration files, among others.

— **Experimental setup**: if authors provide a description of the experimental setup, such as hardware configuration of physical machines, software or systems used, network configurations, among others.

— **Access to results**: if the computed experimental results are available in a public repository, such as: log files, files metric collected during runtime, monitoring data, code to plot charts, among others.

Figure 2.1 summarizes the support for the reproducibility of experiments provided by the selected studies. Regarding the **access to artifacs**, 68% of papers do not provide access to them, and just 24% partially provide (a few artifacts, but not all). Analyzing the description of the **experimental setup**, 76% of papers describe it in detail in a dedicated section of the paper, while 21% only partially describe it, and just 3% do not provide enough information. Lastly,

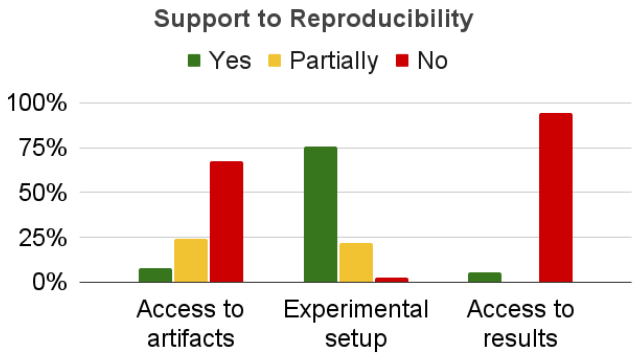


Figure 2.1 – Support to the reproducibility of experiments provided by the selected studies.

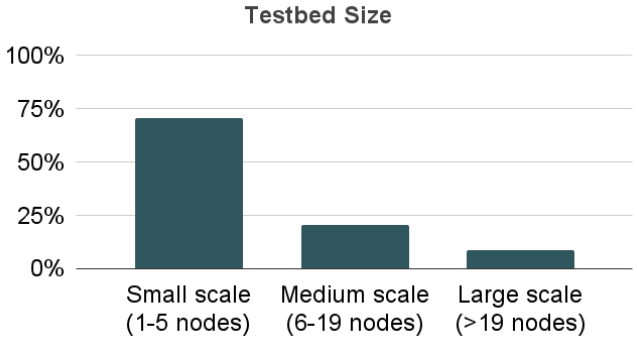


Figure 2.2 – Testbed size used in the experimental evaluations: small scale [244, 126, 46, 135, 243, 110, 222, 61, 189, 63, 99, 245, 146, 7, 94, 6, 236, 101, 129, 83, 125, 66, 20, 147], medium scale [62, 157, 188, 190, 60, 90, 232], and large scale [107, 177, 182]

regarding the **access to results**, 95% of the articles do not provide access, and just 5% provide a public repository with the results. In general, we notice a lack of support for experimental reproducibility in the domain of Edge-to-Cloud experimental research.

Lastly, Figure 2.2 presents the size of the testbeds used in the experimental evaluations. As one may note, 70% of papers use small-scale setups composed of at most five machines or devices, while 20% of them use testbed setups composed of 6 to 19 nodes, and just 10% experiment in large-scale setups with 20 nodes or more.

In conclusion, the results presented in Figure 2.1 reinforce the need for rigorous experimental methodologies that provide guidelines for the reproducibility of experiments in the Edge-to-Cloud research domain. At the same time, Figure 2.2 highlights the need for methodologies and deployment systems guiding researchers to evaluate and validate their proposed approaches in large-scale environments.

2.1.2 Edge-to-Cloud Computing Continuum

The Computing Continuum is a digital infrastructure jointly used by complex application workflows typically combining real-time data generation, processing, and computation. It may include computing resources at central locations such as Cloud datacenters or supercomputers, devices at the Edge, and intermediate infrastructure such as Fog systems.

Edge infrastructures. Refers to computing and storage resources located where the data originated. They consist of potentially many (*e.g.,* millions of) smart devices sensing *"What"* is happening in the environment and generating potentially massive data streams at potentially high rates [87]. Dedicated systems like Apache Nifi [12] push intelligence from the Cloud to those devices and extract value from data in real-time (*e.g.,* improving response times from seconds to milliseconds [100] compared to Cloud-based approaches), while preserving privacy and security (critical data is analyzed locally and not sent remotely).

Fog infrastructures. Refers to a potentially large number of geographically-distributed resources located on the data path between the Edge and the Cloud. Examples include thousands of nodes such as gateways, antennas, routers, and servers [111]. They can be used for in-transit processing on data aggregated from multiple neighboring Edge devices as a way to reduce further data volumes that need to be transferred and processed on Clouds. Lightweight frameworks typically based on message brokers (like Eclipse Mosquitto [77]) that implement the MQTT protocol [145] enable hierarchical processing and intelligent aggregation, minimizing latency and bandwidth usage.

Cloud infrastructures. Provides virtually "unlimited" computing and storage resources used essentially for backup and data analytics for global insight extraction in a centralized way (in its datacenters). Data is first ingested at high rates through dedicated systems (such as Kafka [11], Pulsar [13], ZeroMQ [241], *etc.*) and then analyzed by Big Data processing frameworks (such as Flink [44], Spark [239], Storm [14], among others). They perform stream and batch analytics on vast historical data (in the order of Petabytes), AI model training, and complex simulations [137]. The goal is to help understand *"Why"* the phenomena sensed at the Edge are happening.

2.1.3 Meaningful Experiments on the Computing Continuum

Let us illustrate with a real-life use-case the settings, parameters, and metrics that need to be considered when setting up an experimental Computing Continuum testbed. The application is a Smart Surveillance System [199] which relies on resources from the Edge-to-Cloud Continuum to periodically identify the most crowded areas in a public space, as depicted in Figure

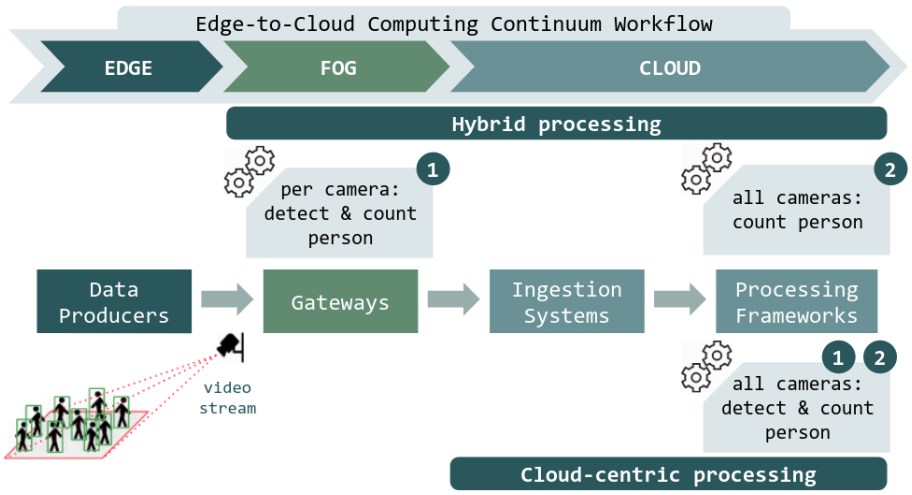


Figure 2.3 – The Smart Surveillance System workflow on the Edge-to-Cloud Continuum.

2.3. The video data processing consists of two phases: *1*) detecting and counting, for each frame of a camera, the maximum number of persons; then *2*) aggregating the maximum number of persons from all geographically distributed cameras to identify the most crowded area.

The question to answer is: *where on this continuum should the video processing take place?* The choices are between a Cloud-centric approach where both phases are executed on the Cloud (bottom of Figure 2.3) and a hybrid approach where processing is split between the Fog and the Cloud (top of Figure 2.3). To address this question, one must explore a solution space in several dimensions: environment settings, configuration parameters, and performance metrics.

The Surveillance System consists of *data producers* such as cameras placed on the Edge; *gateways* in the Fog that receive per area video recordings from Edge devices, provide services like basic data processing and analytics, and forward the data to the Cloud; lastly, *ingestion systems* in the Cloud collect the video streams from all producers and push them to *processing frameworks*. These interconnected components define the experimental environment and workflow.

These components consist of various hardware and software with different constraints and **configuration parameters** that determine their actuation and scenario behavior. For instance, the frequency of video frames on the producers, the streaming window on gateways, or the reserved memory on processing frameworks impacts the workload and the end-to-end performance. Furthermore, the interconnection capabilities vary between the Edge, Fog, and Cloud due to the characteristics of those networks.

Performance metrics of interest in this context are *Fog to Cloud latency*: the time required to send the data from gateways to ingestion systems; *Fog to Cloud throughput*: the amount of data per second that arrives in the ingestion systems; and *end-to-end throughput*: the rate of processed data sent by the processing framework. Besides, *bandwidth, energy, CPU* and *memory usage* help

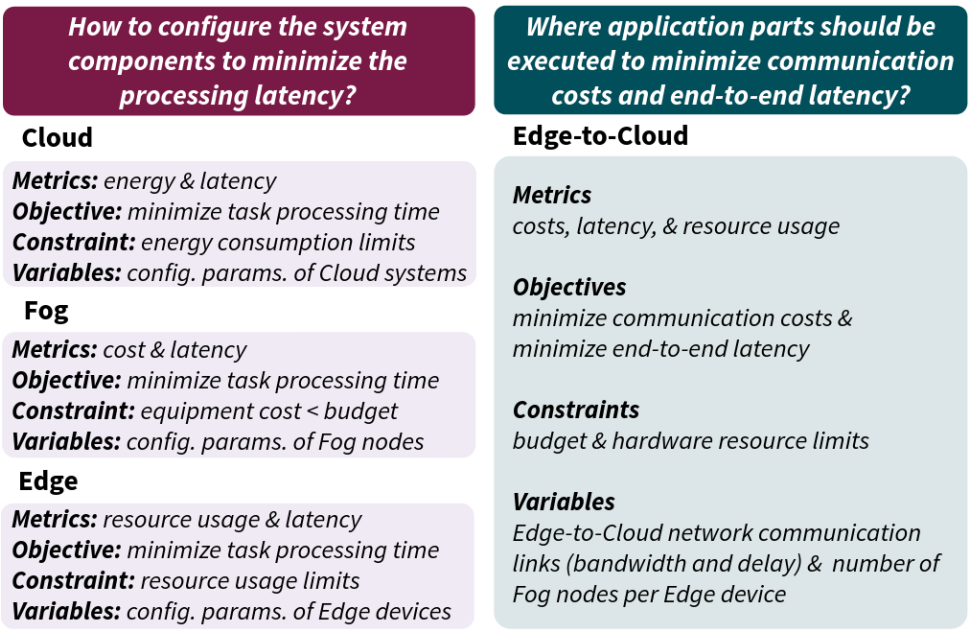


Figure 2.4 – Edge-to-Cloud Continuum optimization problems.

assess the footprint of different design choices.

Formalizing deployment optimization on the Edge-to-Cloud Continuum. Typically, Edge-to-Cloud workflow optimization problems aim at optimizing metrics [29, 19] related to performance (*e.g.*, execution time, latency, and throughput), resource usage (*e.g.*, GPU, CPU, memory, storage, and network), energy consumption, financial costs, and quality attributes (*e.g.*, reliability, security, and privacy). Equation 2.1 describes the formulation of an optimization problem and its mathematical representation: the **optimization variables**, the **objective function**, and the **constraints**.

$$
\begin{aligned}
\min_{x}/\max \quad & f_m(x), && m = 1, 2, \ldots, M \\
\text{subject to} \quad & g_j(x) \leq 0, && j = 1, 2, \ldots, J \quad \textit{Inequality constraints.} \\
& h_k(x) = 0, && k = 1, 2, \ldots, K \quad \textit{Equality constraints.} \\
& x_i^L \leq x_i \leq x_i^U, && i = 1, 2, \ldots, I \quad \textit{Bounds on variables.}
\end{aligned}
\tag{2.1}
$$

The **optimization variables** x refer to the variables associated with the optimization problem (*e.g.*, storage capacity of Edge devices, or number of cores on Fog nodes).

The **objective function** refers to the optimization objective, such as minimizing or maximizing a given metric or set of metrics (*e.g.*, performance, energy consumption). The objective function maps the values of the optimization variables onto real numbers and may be classified as single-objective (such as minimizing Edge-to-Cloud processing latency) or multi-objective

(*e.g.*, minimizing energy consumption of Fog nodes and maximizing throughput).

Finally, the **constraints** refer to requirements that a given solution must satisfy. Constraints may refer to a specific optimization variable (*e.g.*, number of cores on Fog nodes between 10 and 20) and the metrics to be optimized by the objective function (*e.g.*, the maximum response time must be less than 3 seconds).

Figure 2.4 depicts some examples of optimization problems. Left, one would like to answer the question: *how to configure the system components to minimize processing latency?* To reduce complexity, the optimization problem is divided into three sub-problems, each with the objective of minimizing the task processing time on the Edge, Fog, and Cloud infrastructures, under specific constraints. The right-hand example aims at answering the question: *where should the workflow components be executed to minimize communication costs and end-to-end latency?* This translates into a single multi-objective optimization problem (minimizing communication costs and end-to-end latency), as opposed to the previous example (several single-objective optimization problems).

In order to model and solve such optimization problems, one may find multiple methods in the literature. For instance, packages and libraries such as *Scikit-Optimize* [194], *Scikit-Learn* [156], *Surrogate Modeling Toolbox* (*SMT*) [36], *DeepHyper* [24], *etc.*, may be used to build surrogate models and then use those model to explore the search space of the optimization problem.

2.2 Existing Systems for Analyzing Edge-to-Cloud Workflows

Table 2.2 summarizes the main open-source state-of-the-art simulation, emulation, and deployment systems for experimental research on the Edge-to-Cloud Continuum. Furthermore, it presents workflow management systems supporting Edge-to-Cloud applications.

2.2.1 Simulation, Emulation and Deployment Systems

Building experimental testbed environments is expensive and brings challenges to conducting reproducible experiments. In this sense, simulation systems play an important role as they allow to analyze systems behavior at a very large scale while easily tuning a myriad of configuration parameters. Next, we present simulation systems used in the modeling of Cloud, Fog, and Edge computing environments.

Simulation Systems

Cloud-based simulation systems. CloudSim [43] framework allows modeling and simulation of Cloud computing infrastructures and services in a repeatable manner. CloudSim allows users to model the behavior of data centers, Virtual Machines, and resource provisioning poli-

cies. ElasticSim [41] is a workflow simulator that extends CloudSim. It focuses on supporting resource runtime auto-scaling and stochastic task execution time modeling. SCORE [85] allows the execution of heterogeneous workloads for simulating energy-efficient monolithic and parallel-scheduling models.

Fog-based simulation systems. FogExplorer [106] provides modeling and simulation to estimate QoS and cost evaluation of Fog-based IoT applications. FogExplorer allows users to choose good application designs during its design phase. FogTorch [40] aims to support the deployment of IoT applications in Fog infrastructures considering software, hardware, and QoS requirements. FogNetSim++ [165] focuses on simulating large Fog networks and differs from others mainly by providing features that allow users to incorporate customized mobility models, scheduling algorithms, and manage handover mechanisms. XFogSim [138] extends FogNetSim++ to simulate federated Fog computing environments. xFogSim is lightweight, configurable, and scalable, and introduces the concept of Fog federation for resource sharing among Fog locations. Furthermore, it allows users to evaluate applications in terms of energy consumption, processing latency, scalability, and resource usage. YAFS [128] aims to allow users to analyze application designs and incorporate strategies for placement, scheduling, and routing. YAFS also supports the dynamic allocation of new application modules, dynamic failures of network nodes, and user mobility. Furthermore, it facilitates the shareability of experiment results by generating logs of workload generation and computation, and link transmissions. Lastly, iFogSim [102] focuses on resource management techniques in IoT, Edge, and Fog computing environments. iFogSim allows users to measure, in a repeatable manner, the impact of resource management techniques in terms of latency, network congestion, energy consumption, and cost.

Edge-based simulation systems. EdgeCloudSim [205] focuses on Edge Computing scenarios and allows one to conduct experiments considering computational and networking resources. IoTSim-Edge [148] allows users to easily configure their Edge infrastructures and to capture the behavior of heterogeneous IoT and Edge devices in terms of sensing, processing, mobility, and data rate. Both Edge systems extend CloudSim.

Emulation Systems

Compared to simulation, the emulation approach provides more realistic results. While simulators mimic the behavior and configurations of a real device, emulation systems duplicate the hardware and software features of a real device [209]. Emulation systems are also a less expensive solution when compared to real deployments.

Fogbed [55] allows resource provisioning emulation in Fog environments. It combines Containernet [158] and Maxinet [226] (both are extensions of the Mininet [117] network emulator)

to allow the use of virtual instances for resource provisioning emulation.

EmuFog [139] focuses on the design of Fog Computing infrastructures and the emulation of real applications and workloads. In EmuFog, users can: design the network topology; embed Fog nodes in the topology; and run Docker-based applications on those nodes connected by an emulated network.

RADICAL-DREAMER [166] provides the concepts of *Task* and *Workload* to model the characteristics of an application according to heterogeneous tasks. Besides, it provides the concept of *Resource* to model distributed infrastructures. RADICAL-DREAMER allows users to evaluate deployment configurations, performance trade-offs, and workload placement strategies for Edge-to-Cloud applications [136].

Deployment Systems

Deploying real-life applications on large-scale testbeds provides the most realistic results compared to simulation or emulation approaches. In this direction, a few systems have been proposed in the past few years.

Kubernetes [38] aims to simplify the deployment and management of services that compose an application by providing mechanisms for deployment, maintenance, and scaling. Using Kubernetes, users can manage containerized applications across multiple hosts.

KubeEdge [233] builds on top of Kubernetes to extend Cloud capabilities to the Edge and allows containerized application orchestration and device management to hosts at the Edge. KubeEdge key features are core infrastructure support for networking, application deployment, and metadata synchronization between Cloud and Edge.

2.2.2 Workflow Management Systems

Scientific workflows allow users to execute computational tasks typically composed of multiple steps. Recently, such workflows have started to be deployed on distributed and heterogeneous Edge-to-Cloud computing infrastructures. Hence, workflow management systems have been extended to allow the execution of workflows on highly distributed resources from IoT sensors and devices in the Edge to HPC and Cloud computing resources.

Pegasus [64] workflow management system allows users to map high-level workflow descriptions onto distributed resources such as clusters, grids, and clouds. It abstracts all the complexities of deploying workflows on the underlying execution environment. Pegasus has been extended to support Edge-to-Cloud workflows [215] (*e.g.*, executed on the Chameleon Cloud and Edge testbeds). This extension aims to allow scientists to explore performance trade-offs in managing and executing Edge-to-Cloud workloads. Results show that Pegasus helps scientists to conduct Edge-to-Cloud research. Table 2.2 summarizes the key features.

COMPSs [21] is a task-based programming model which aims to ease the development and execution of parallel applications (written in Java, Python or C/C++) in distributed computing infrastructures (*e.g.*, clusters, clouds, *etc.*). COMPSs applications are composed of tasks that may be annotated with different constraints in terms of: computing resources such as GPU, number of cores, *etc.*; memory available; among others. Such applications are agnostic of the underlying computing infrastructure. In [22], authors extended COMPSs to manage distribution, parallelism, and heterogeneity in Edge resources transparently to the application programmer.

EdgeWorkflow [235] is a workflow management system that allows users to deploy and manage workflows in Edge computing environments. It supports the creation of Edge environments according to user settings, as well as the automatic deployment, monitoring, and performance evaluation of workflows executed on the Edge. Evaluations show how EdgeWorkflow helps to assess the performance of workflows regarding the energy consumption of the end device, execution cost, and execution time.

Delta [127] is a service for scheduling function executions (Function as Service - FaaS paradigm) across heterogeneous and distributed resources from Edge devices to Clouds and supercomputers. Through predictors and scheduling algorithms (*e.g.,* for function runtime, data transfer time, configuration delay, and cold start resource provisioning), Delta provides dynamic estimates of function execution times to determine the most appropriate location for execution. Experiments show that Delta can halve workload makespan when compared with existing strategies. In the same direction, authors propose OpenWolf [197], a serverless workflow management system that explores the FaaS paradigm for composing complex workflows on the Cloud-Edge Continuum. OpenWolf allows users to describe the workflow structure in terms of processes, relationships, used functions, and scheduling constraints.

As presented, the existing systems focus on the Edge, Fog, or Cloud infrastructures. **Very few** aim to support experimentation on the **whole Computing Continuum**, from the IoT/Edge to the Fog and Cloud/HPC infrastructures.
Therefore, enabling the Computing Continuum vision r**equires the development of novel systems** (beyond the ones presented in Table 2.2) **or advancing existing systems** like Pegasus [64], COMPSs [55], or OpenWolf [197] for addressing the requirements of Edge-to-Cloud experiments. Such systems, for enabling reproducibility, should abstract the complexities of deploying Edge-to-Cloud workflows on open scientific testbeds for large-scale evaluations.
In addition, they should provide support to the management of the whole experimental cycle such as monitoring, the gathering of results, provenance capture, performance optimization, and repeating experiments on the same infrastructure are extremely relevant.

2.3 Large-scale testbeds for Edge-to-Cloud Experiments

Several experimental testbeds allow researchers to evaluate their proposals in real-life settings by providing access to a large number of resources (grouped in homogeneous or heterogeneous clusters, upon convenience) and, more importantly, supported by some vibrant communities of users and solid technical teams. We cite here just a few.

2.3.1 Testbeds Explored in this Work

Grid'5000 [31] is a large-scale French testbed for experimental research with a focus on parallel and distributed computing including Cloud, HPC, Big Data, and AI. Grid'5000 is merging with FIT IoT-Lab to enable Edge-to-Cloud experiments. **FIT IoT-Lab** [4] is a large-scale multi-radio (*e.g.,* IEEE 802.15.4, Bluetooth Low Energy, LoRa, *etc.*) and multi-platform (*e.g.,* Arduino Zero, nRF52840-MDK, LoRa gateway, and many others) infrastructure for the Internet of Things. FIT IoT-Lab consists of more than 1.5K nodes and provides tools for monitoring energy consumption and network-related metrics, such as end-to-end delay, throughput, and overhead.

Chameleon [120] is a large-scale US experimental platform that aims to support Computer Science research in many areas, such as systems, storage, networking, GPU, security, Artificial Intelligence, and High-Performance Computing. **CHI@Edge** is an extension of the Chameleon testbed that aims to support Edge Computing experiments. Combining Chameleon and CHI@Edge testbeds allows more realistic Edge-to-Cloud experiments since it provides access to real-life IoT/Edge devices such as Raspberry Pis, Jetson Nanos, among others.

2.3.2 Other Relevant Testbeds

ORBIT [151] (Open-Access Research Testbed for Next-Generation Wireless Networks) is based on a 20x20 two-dimensional grid of programmable radio nodes which can be interconnected into different topologies. ORBIT provides access to radio resources, including WiFi 802.11 a/b/g 802.11n 802.11ac, Bluetooth (BLE), ZigBee, and Software Defined Radio platforms; Software-defined networking (SDN) resources; LTE and WiMAX base stations and clients; and Cloud resources such as nodes with Tesla-based GPUs.

SmartSantander [187] is a large-scale testbed composed of around 2000 IEEE 802.15.4 devices deployed in a 3-tiered architecture (IoT node, repeaters, and gateway node) deployment in the Spanish city of Santander. The testbed allows IoT native experimentation (e.g. wireless sensor network experiments) and service provision experiments (e.g. applications using real-time real-world sensor data).

Fed4FIRE+ [65] is a project offering the largest federation worldwide of Next Generation Internet (NGI) testbeds. Fed4FIRE aims to provide open, accessible, and reliable experimental

infrastructures supporting a wide variety of research, such as 5G, IoT, Cloud Computing, and wired and wireless Computer Networking. The list of testbeds [149] federated with Fed4FIRE are: CityLab [213], PlanetLab [86], ExoGENI [26], Tengu [220], NITOS [155], w-iLab [35], among others.

2.4 Open Challenges Explored in this Work

As presented in the previous chapter, distributed infrastructures for Data Analytics and learning are now evolving towards an interconnected ecosystem allowing complex applications to be executed from IoT Edge devices to the HPC Cloud. Next, we present some of the relevant challenges and research opportunities to be addressed to enable the Computing Continuum vision. All the findings presented in this chapter are published in a systematic review [175].

2.4.1 Understanding Performance of Edge-to-Cloud Workflows

Understanding end-to-end performance on the complex Edge-to-Cloud Continuum ecosystem is challenging. Deploying large-scale real-life applications on such infrastructures requires configuring a myriad of system-specific parameters and reconciling many requirements or constraints in terms of hardware capacity, mobility, network efficiency, energy, and data privacy, with low-level infrastructure design choices. One important challenge is to accurately reproduce relevant behaviors of a given application workflow and representative settings of the physical infrastructure underlying this complex continuum.

A first step towards reducing this complexity and enabling the Computing Continuum vision is to enable a holistic understanding of performance in such environments. That is, finding a rigorous approach to answering questions like (*1*) *How to identify infrastructure bottlenecks across the whole Edge-to-Cloud Continuum?* (*2*) *Which system parameters and network configurations impact the application performance and how?* (*3*) *How Edge-to-Cloud hardware configurations impact on the energy consumption and on the processing latency of the application?*

Approaches based on workflow modeling [186] and simulation or emulation, as presented in Table 2.2, raise some important challenges in terms of specification, modeling, and validation in the context of the Computing Continuum [3, 214]. For example, it is increasingly difficult to model the heterogeneity and volatility of Edge devices or to assess the impact of the inherent complexity of hybrid Edge-Cloud deployments on performance. At this stage, experimental evaluation remains the main approach to gain accurate insights on performance metrics and to build precise approximations of the expected behavior of large-scale applications on the Computing Continuum, as a first step prior to modeling.

A key challenge in this context is to be able to **reproduce in a representative way the application behavior in a controlled environment**, for extensive experiments in a large enough

spectrum of potential configurations of the underlying hybrid Edge-Cloud infrastructure. However, this process is non-trivial due to the multiple combination possibilities of heterogeneous hardware and software resources, as well as, system components for Data Analytics and Machine Learning. Therefore, the Computing Continuum vision calls for novel approaches to map the real-world application components and dependencies to infrastructure resources.

Further research efforts shall necessarily focus on the design and implementation of novel methodologies and systems for large-scale experimental evaluation covering the characteristics of hybrid Edge-Cloud infrastructure deployments. Novel systems allowing the combination of simulation and emulation systems in addition to supporting the deployment of state-of-the-art systems for Data Analytics and Machine Learning on real-world large-scale testbeds, considering the same experimental evaluation package, would be relevant to accurately reproducing complex application behaviors.

2.4.2 Optimizing Performance of Edge-to-Cloud Workflows

The optimization of application workflows on highly distributed and heterogeneous resources is challenging. Real-world applications deployed on hybrid Edge-to-Cloud infrastructures (*e.g.,* smart factory [223], autonomous vehicles [142], among others) typically need to comply with many conflicting constraints related to hardware resource consumption (*e.g.,* GPU memory, CPU power, main memory size, storage size, and bandwidth), software components composing the application and requirements such as QoS, security, and privacy [231].

Furthermore, Edge-to-Cloud deployment optimization problems aim at optimizing metrics [29, 19] related to performance (*e.g.,* execution time, latency, and throughput), resource usage, energy consumption, financial costs, and quality attributes (*e.g.,* reliability, security, and privacy). The parameter settings of the applications and the underlying infrastructure result in a complex multi-infrastructure configuration search space [167].

Therefore, one important challenge is to accurately and efficiently answer questions like (*1*) *How to configure the hardware and system components to minimize processing latency and energy consumption?* (*2*) *Where should the workflow components be executed across the Edge-to-Cloud Continuum to minimize communication costs and end-to-end latency?* (*3*) *How to efficiently autoscale the application resources concerning workload fluctuations and infrastructure changes?*

Such optimization problems are of NP-hard complexity and multi-objective [39, 123]. Furthermore, the environment settings and configuration parameters are extremely vast, and their combination of possibilities is virtually unlimited [196, 234]. Hence, the process of searching for the ideal deployment and configuration of those real-life applications is challenging given the search space complexity: bad choices may result in increased financial expenses during deployment and production phases, decreased processing efficiency, and poor user experience [221].

Given these complexities, future research should focus on proposing novel optimization

methodologies supporting the parallel deployment and evaluation of such complex application workflows on real-life large-scale testbeds. The objective is twofold: speeding up the optimization computations, as well as obtaining more accurate results.

Novel approaches should also rely on the development of fully automated surrogate model building to mimic and approximate the complex behavior of Edge-to-Cloud workflows and then perform optimization and sensitivity analysis. These new solutions may combine computationally tractable optimization techniques [159] such as Bayesian Optimization [204] methods (*e.g.,* Gaussian process (Kriging) [203], Decision Trees [224], Random Forest [37], among others) to build surrogate models; and then combine with techniques such as evolutionary algorithms and swarm intelligence based algorithms (*e.g.,* Genetic Algorithm [143], Differential Evolution [59], Particle Swarm Optimization [73], *etc.*) to perform and to speed up the optimization (*e.g.,* to find the optimal deployment configuration using the built surrogate model).

Novel contributions are required for workload characterization and prediction, for auto-scaling strategies to enable the efficient scaling of distributed application resources across the Edge-to-Cloud Continuum, in response to workload fluctuations and infrastructure changes. Contributions in this context should be aligned with the complex heterogeneous characteristics of the Computing Continuum paradigm, in terms of computing resources, network constraints, and application requirements.

2.4.3 Enabling Reproducible Analysis of Edge-to-Cloud Workflows

Given the relevance of experimental reproducibility in scientific research to allow the verification of the scientific claims and also to evolve the studies, in addition to the lack of support for the reproducibility of experiments identified in recent articles, as presented in Section 2.1, future research efforts should focus on the design and implementation of rigorous methodologies for experimental reproducibility.

Supporting the reproducibility of experiments carried out on large-scale distributed and heterogeneous infrastructures is non-trivial. The experimental methodology, the artifacts used, and the data captured should provide additional context that more accurately explains the experiment execution and results.

One relevant challenge is to provide mechanisms to allow researchers to **repeat**, **replicate**, and **reproduce** the scientific claims and to help them answer questions like (*1*) *What machines/devices were used to execute the entire workflow?* (*2*) *What steps were invoked during the workflow execution?* (*3*) *Which infrastructure configurations and application parameters produced these results?*

Therefore, novel approaches should focus on enabling the repeatability, replicability, and reproducibility of experiments. This requires the **definition of rigorous experimentation methodologies** (*e.g.,* well-defined description of hardware and software resources required to run the experiments and their configurations, network setups, resource interconnections, and workflow

execution logic); the **access to the experimental artifacts** (*e.g.*, datasets, scripts, libraries, applications, systems, configuration files, among others); and the **access to results** (*e.g.*, log files, metrics collected during execution, monitoring data, code to plot results, among others).

Answering the above questions may be assisted by **provenance data capture**. Capturing provenance data of Edge-to-Cloud workflows requires the design and development of novel approaches. The efficient data capture (*e.g.*, low capture overheads in terms of CPU and memory usage, network usage, and power consumption) on heterogeneous hardware resources ranging from HPC/Cloud servers to resource-constrained IoT/Edge devices remains an open issue.

Conclusion

The results presented in Figure 2.1 reinforce **the need for rigorous experimental methodologies** guiding researchers to systematically perform Edge-to-Cloud experiments to enable the experiment **reproducibility**. At the same time, Figure 2.2 highlights **the need for systems guiding researchers** to **deploy**, **evaluate**, and **validate** their approaches in **large-scale testbeds**.
Furthermore, realizing the Computing Continuum vision in practice raises many research questions. Therefore, novel approaches should focus on **addressing challenges** such as **understanding** the end-to-end **performance** of application workflows, enabling their **optimized execution** across the Edge-to-Cloud Continuum, and systematically performing experiments to **enable their reproducibility and replicability** by independent researchers. These challenges are explored in this thesis and presented in the next chapters.

Table 2.2 – Simulation, Emulation, and Deployment Systems for Experimental Research on the Edge-to-Cloud Continuum.

Simulation Systems	Edge	Fog	Cloud	Main Goal	Key Features
CloudSim [43]			✓	Modeling, simulation, and experimentation of Cloud infrastructures and application services.	— modeling and simulation of large-scale Cloud computing data centers. — modeling and simulation of virtualized server hosts and application containers. — modeling and simulation of energy-aware computational resources. — modeling and simulation of data center network topologies.
SCORE [85]			✓	Simulate energy-efficiency, security, and scheduling strategies in Cloud Computing environments.	— allows to prototype and compare different cluster scheduling strategies. — generates synthetic workloads from empirical parameter distributions. — allows the analysis of scheduling performance metrics.
ElasticSim [41]			✓	Simulate autoscaling algorithms.	— supports resource auto-scaling. — supports stochastic task execution time modeling.
iFogSim [102]		✓		Modeling and simulation of resource management techniques in IoT, Edge, and Fog Computing environments	— inherits features from CloudSim. — provides resource management techniques in IoT, Edge, and Fog. — allows simultaneous execution of applications on the infrastructure. — supports migration of application modules from one Fog device to another.
FogNetSim [165		✓		Simulate distributed Fog computing environments.	— covers the network aspects such as delay, packet error rate, transmission range, handover, scheduling, and heterogeneous mobile devices. — allows to simulate a large Fog network. — allows to simulate heterogeneous devices with varying features. — supports handover: allows static and dynamic nodes in the network.

Simulation Systems	Edge	Fog	Cloud	Main Goal	Key Features
FogTorch [40]		✓		QoS-aware deployment of IoT applications through the Fog.	— allows the specification of a Fog infrastructure along processing (*e.g.*, CPU cores, RAM memory, storage) and QoS (*e.g.*, latency, bandwidth) capabilities. — allows applications to be deployed along with needed IoT devices, processing and QoS requirements.
FogExplorer [106]		✓		Simulate QoS and cost evaluation of Fog-based IoT applications.	— simulates processing cost and processing time for applications. — simulates transmission cost and transmission time for data streams.
IoTSim-Edge [148]	✓			Simulate the distribution and processing of streaming data generated by IoT devices in Edge computing environments.	— allows to define data analytic operations and their mapping to different parts of the infrastructure. — supports modeling of heterogeneous IoT protocols along with their energy consumption profile. — supports modeling of mobile devices and captures the effect of handoff caused by the movement of mobile devices.
EdgeCloud-Sim [205]	✓			Simulate environments specific to Edge Computing scenarios.	— considers computing and networking resources. — supports network modeling specific to WLAN and WAN. — supports device mobility model and provides tunable load generator.
YAFS [128]	✓			Analyze the design and deployment of applications through customized and dynamical strategies.	— allows dynamic scenarios: placement, path routing, orchestration, and workload movement. — supports placement allocation algorithms and orchestration algorithms. — provides functions to obtain metrics such as network utilization, network delay, response time, and waiting time.
XFogSim [138]	✓			Simulate federated Fog computing environments.	— provides resource allocation algorithms for resource sharing. — supports static and mobile nodes (handover mechanisms). — supports evaluations in terms of energy consumption, processing latency, scalability, and resource usage.

Emulation Systems	Edge	Fog	Cloud	Main Goal	Key Features
EmuFog [139]		✓		Enable the design of Fog Computing infrastructures and the emulation of real applications and workloads.	— generates networks that can be emulated easily with MaxiNet [226]. — supports topologies from BRITE [140] and Caida [42]. — places Fog nodes based on user-defined constrains (*e.g.*, network latency or resource constraints).
Fogbed [55]		✓		Enable the rapid prototyping of Fog components in virtualized environments.	— allows dynamic topology changes. — provides traffic control links such as delay, rate, loss, and jitter. — enables the deployment of Fog nodes as software containers under different network configurations.
RADICAL-DREAMER [166]	✓	✓	✓	Emulate resource and task/-workload definition in Edge-to-Cloud applications.	— allows to evaluate workload and resource management aspects of applications. — supports modeling task placement in Edge-to-Cloud applications. — allows to evaluate deployment modalities and performance trade-offs.
Deployment Systems	**Edge**	**Fog**	**Cloud**	**Main Goal**	**Key Features**
KubeEdge [233]	✓			Deploy complex high-level applications to the Edge.	— provides containerized application orchestration and device management to hosts at the Edge. — provides core infrastructure support for networking, application deployment, and metadata synchronization between Cloud and Edge. — supports MQTT which enables Edge devices to access through Edge nodes.
Kubernetes [38]			✓	Manage and automate the deployment, scaling, and management of containerized applications across multiple hosts.	— provides mechanisms for deployment, maintenance, and scaling of applications. — provides service discovery and load balancing. — allows to automatically mount storage systems, such as local storage and public Cloud providers.

Workflow Management Systems	Edge	Fog	Cloud	Main Goal	Key Features
Pegasus [64]	✓	✓	✓	Develop, run, monitor, and debug large-scale scientific workflows.	— allows workflows to be reused in different environments. — prioritizes tasks in order to increase workflow performance. — provenance data collection to capture performance metrics to optimize workflow executions. — scales the size of the workflow and its resources. — provides debugging and recovery strategies.
COMPSs [55]	✓	✓	✓	Ease the development and execution of applications on distributed infrastructures.	— supports applications written in Java, C/C++, and Python. — applications are agnostic of the computing infrastructure. — allows expressing resource constraints on tasks. — provides task monitoring at runtime.
EdgeWork -flow [233]	✓			Deploy and manage workflows in Edge Computing environment.	— allows users to define Edge environments. — provides deployment and monitoring of workflows. — supports workflow performance metrics like execution time, energy consumption, and cost.
Delta [166]	✓	✓	✓	Schedule function-based workloads across Edge-to-Cloud resources.	— provides profiling and predicting function performance. — manages data movements across resources. — routes function executions to resources based on scheduling policies.
OpenWolf [38]	✓	✓	✓	Explore FaaS paradigm for composing workflows across the Cloud-Edge Continuum.	— supports relationships between functions. — supports parallel process execution. — allows data pre/post-processing filters.

Part II

E2C*lab*: Exploring the Computing Continuum through Repeatable, Replicable and Reproducible Edge-to-Cloud Experiments

Chapter 3

Our Methodology

Contents

The second part of the thesis focuses on addressing the challenges of understating and optimizing the performance of Edge-to-Cloud workflows through repeatable, replicable, and reproducible experiments on open scientific testbeds. To do so, we introduce novel methodologies and their implementation within frameworks and tools.

This Chapter introduces our experiment methodology that guides researchers to deploy Edge-to-Cloud workflows on large-scale testbeds to understand application performance. Then, it presents our optimization methodology for optimizing the performance of workflows executed across the Computing Continuum.

3.1 The Need for Rigorous Experiment Methodologies

The Computing Continuum vision calls for a rigorous and systematic methodology to map the real-world application components and dependencies to infrastructure resources. As illustrated in Figure 3.1, this complex process can be error-prone. Key research goals are *1*) to identify relevant characteristics of the application workflows and of the underlying infrastructure as a means to enable accurate experimentation and benchmarking in relevant infrastructure settings in order to understand and optimize their performance, and *2*) to ensure research quality aspects such as the 3Rs.

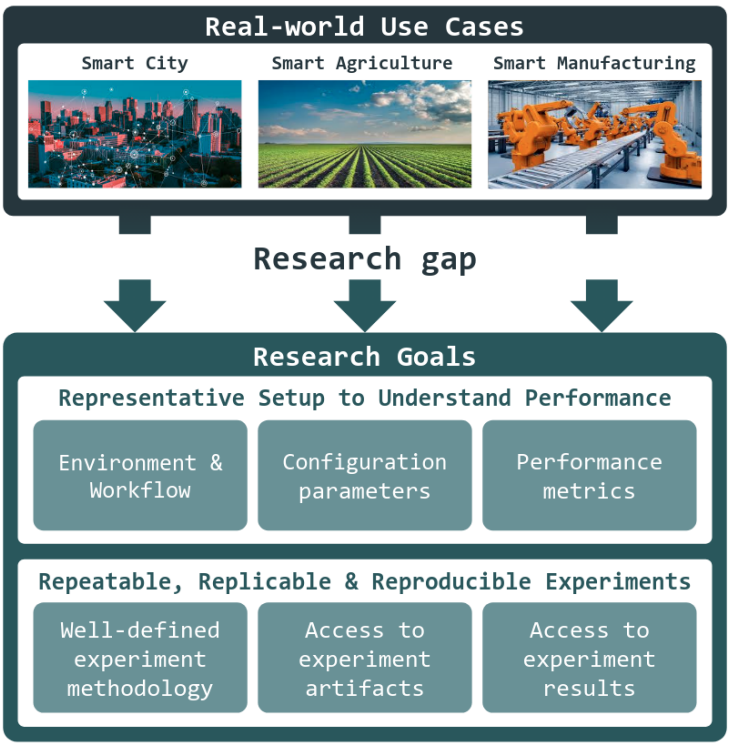


Figure 3.1 – Enabling representative 3Rs experiments of real-world use cases on the Computing Continuum.

The main contributions of our approach are:

1. **A rigorous methodology for designing experiments with real-world workflows on the Computing Continuum** spanning from the Edge through the Fog to the Cloud; this methodology supports Open Science [150] and provides guidelines to move from real-world use cases to the design of relevant testbed setups for experiments enabling researchers to understand performance and to ensure the 3Rs properties (Section 3.2);
2. **A methodology to optimize the performance of real-life applications on the Computing Continuum**, leveraging computationally tractable optimization techniques (Section 3.3).
3. **A novel framework named E2Clab** that implements these methodologies and allows researchers to deploy their use cases on real-world large-scale scientific testbeds such as Grid'5000 [31], Chameleon Cloud and CHI@Edge [120], and FIT IoT-LAB [4] (Chapter 4).

   To the best of our knowledge, E2Clab is the first platform to support the complete analysis cycle of an application on the Computing Continuum: (*i*) *Reproducible Experiments:* supports repeatability, replicability, and reproducibility; (*ii*) *Mapping:* application parts (Edge, Fog and Cloud/HPC) and physical testbed; (*iii*) *Variation and Scaling:* experiment variation and transparent scaling of scenarios; (*iv*) *Network Emulation:* Edge-to-Cloud com-

munication constraints; (*v*) *Experiment Management:* deployment, execution and monitoring; and (*vi*) *Optimization:* configuration search of application workflows.

4. **A large scale experimental validation** of the proposed approach with the Pl@ntNet application on 42 nodes of the Grid'5000 testbed [32]. Our approach helps understand and optimize Pl@ntNet software configurations across the continuum to minimize user response time (Chapter 5).

3.2 A Methodology for Designing Reproducible Experiments with Real-life Applications on the Computing Continuum

Our methodology is based on the takeaways of a previous study on Edge and Cloud computing trade-offs [199]. The specific experimental approach used for that study is generalized and defined here in a standalone methodology, which can be used by any application and deployed anywhere on the Computing Continuum. The methodology leverages three main processes which consist of a series of actions to achieve the high-level goals of this thesis. Found at the left side of Figure 3.2 are these processes pipelined in a stream fashion.

3.2.1 Providing Access to Experiment Artifacts

This process, illustrated at the bottom of Figure 3.2, consists in providing access to all the research artifacts used to enable the experiments. They include the original dataset used as input to the experiments; the software, algorithms, libraries, *etc.*, developed by experimenters or obtained from third parties; and the whole experiment configuration details such as the hardware specifications, execution parameters, the network configuration, and the experiment workflow.

In order to enable the 3Rs of experiments, all these research artifacts must be in public and safe repositories.

3.2.2 Defining the Experimental Environment

This process has as its main goal to define the whole experimental environment: the layers, the services, the network interconnect, and the experimental workflow, illustrated as subprocesses at the core of Figure 3.2.

The methodology is centered around the concepts of **Services** and **Layers**:

— **Services** represent any system that provides a specific functionality or action in the scenario workflow. *Services* may refer to data producers, gateways, or processing frameworks (*e.g.,* a Flink cluster). A *service* can be composed of *sub-services*, for instance, a Flink service composed of a Job Manager (*master*) and Task Managers (*workers*).

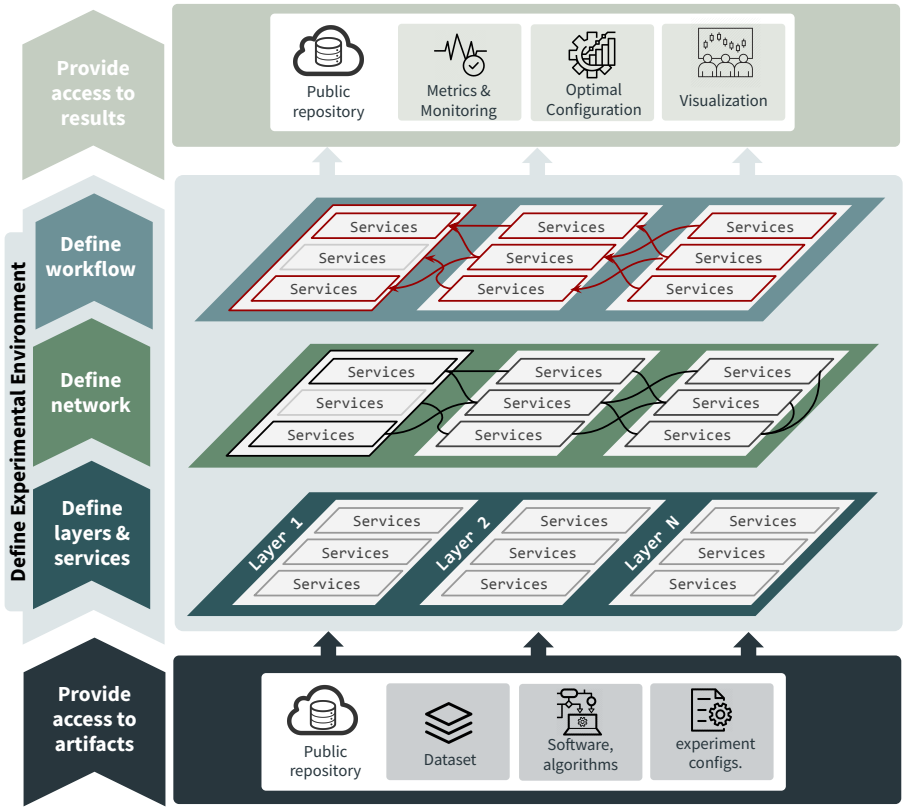


Figure 3.2 – Our experimental methodology.

— **Layers** define the hierarchy between *services* and group them with different granularities. They can also be used to reflect the geographical distribution of the compute resources. In the context of the Computing Continuum, *layers* refer to Edge, Fog, and Cloud, for instance.

This *layer* and *service* abstraction targets experiments scalability and variation, since it allows to easily resize the number of *layers* and to analyze different scenario deployments such as single-layered (*e.g.*, Cloud-only) or multi-layered (*e.g.*, Edge + Cloud). Next, we describe the roles of each sub-process.

1. *Define Layers and Services*: specifies the *layers* and the *services* deployed on each *layer* that compose the experimental scenario. Each *service* must be configured with its specific parameters and low-level configurations according to the experimenters requirements. Each *service* can be monitored (processor, storage, memory, network, I/O, *etc.*) during the execution of the experiments.
2. *Define the Network*: specifies the network communication rules between *layers* and between *services*. For each network communication, the experimenter should be able to define specific network conditions and constraints.

3. *Define the Workflow*: specifies all the execution logic and rules of the software, algorithms, and applications running on *services* (*i.e.,* data producers, ingestion systems, processing engines, *etc.*). Such execution logic and rules refer to dependencies between *services* (*e.g.,* execution order of services and how they are interconnected) and managing their life cycle (*e.g.,* preparing all dependencies required to properly initiate their execution; launching them with their respective parameters; and finalizing them). The workflow also includes supporting components (not included in the experiment analysis) to collect performance metrics, such as throughput, latency, among others.

Note that breaking down the definition of the experimental environment into three well-defined sub-processes enables flexibility and variability, since the definitions of each sub-process may be modified without impacting the remaining ones.

3.2.3 Providing Access to Experiment Results

This process, illustrated at the top of Figure 3.2, collects all the output generated during the execution of the experiments (*i.e.,* log files, monitoring data, performance metrics, *etc.*). These outputs may be aggregated and analyzed according to the experimenters interests to derive insights and understand performance. Lastly, to enable the 3Rs, all the output data, the research results, and the conclusions must be available in a public and safe repository.

3.3 A Methodology for Optimizing the Performance of Real-life Applications on the Edge-to-Cloud Continuum

The former methodology allows for performing reproducible experiments to understand the application performance. Such applications typically need to comply with many constraints related to resource usage (*e.g.,* GPU, CPU, memory, storage, and bandwidth capacities), energy consumption, QoS, security, and privacy [231]. Therefore, enabling their optimized execution across the Edge-to-Cloud Continuum is challenging. The parameter settings of the applications and the underlying infrastructure result in a complex configuration search space [167]. Next, we present our optimization methodology. It supports reproducible parallel optimization of application workflows on large-scale testbeds. It consists of three main phases illustrated in Figure 3.3.

3.3.1 Phase I: Initialization

This phase, depicted at the top of Figure 3.3, consists in defining the optimization problem. The user must specify: the **optimization variables** that compose the search space to be explored (*e.g.,* GPUs used for processing, Fog nodes in the scenario, network bandwidth, *etc.*);

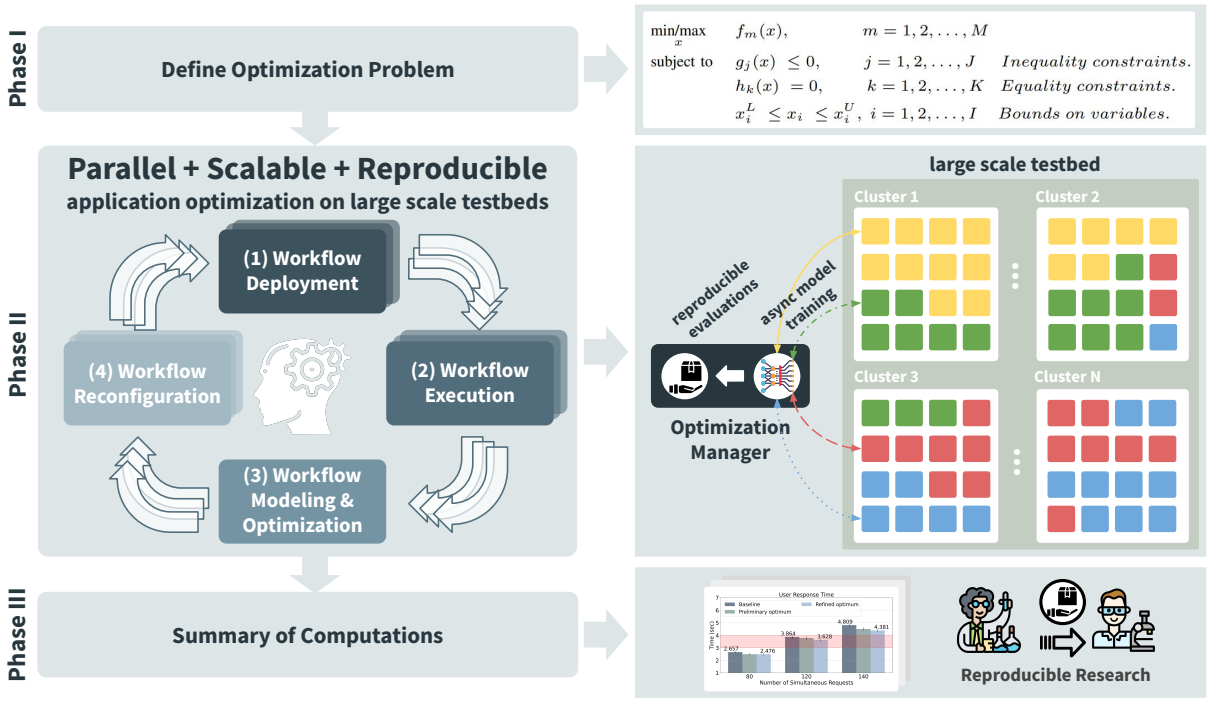


Figure 3.3 – Our optimization methodology.

the **objective** (*e.g.*, minimize end-to-end latency, maximize Fog gateway throughput, *etc.*); and **constraints** (*e.g.* the upper and lower bounds of optimization variables, budget, latency, *etc.*).

One may focus the optimization on: (1) specific parts of the infrastructure (*e.g.*, only on geographically distributed Edge sites, or only on Fog-to-Cloud resources) by defining multiple, per infrastructure, optimization problems. This approach reduces the search space complexity (in case of use cases with large search spaces) and hence the computing time; (2) or the whole Edge-to-Cloud infrastructure as a single optimization problem.

3.3.2 Phase II: Evaluation

This phase aims at defining the mathematical methods and optimization techniques used in the *optimization cycle* (presented in the middle of Figure 3.3) to explore the search space. Such *optimization cycle* consists in: (1) parallel deployments of the application workflow in a large-scale testbed; (2) their simultaneous execution; (3) asynchronous model training and optimization with data obtained from the workflow execution; and (4) reconfiguration of the application workflow for a new evaluation.

This cycle continues until model convergence or after a given number of evaluations defined by the user. Depending on the run-time characteristics of the application workflows, their evaluations may be performed differently.

Long-time Running Applications

These refer to experiments or simulations for which the evaluation of a single point in the search space requires a lot of time to complete (*e.g.*, hours, or even days). Since application workflows in the context of the Computing Continuum typically consist of cross-infrastructure parameter configurations resulting in a myriad of configuration possibilities, their optimization problem presents a complex and large search space.

For those long-time running applications, a variety of Bayesian Optimization [204] methods (*e.g.*, surrogate models as Gaussian process (Kriging) [203], Decision Trees [224], Random Forest [37], Gradient Boosting Regression Trees [89], Support Vector Machine [211], Polynomial Regression [152], among others) may be applied as candidates to explore the search space. Their generation is described below.

Surrogate Model Building: this consists of three steps: (*a*) a few sample points are generated, respecting the upper and lower limits of each optimization variable that composes the search space. Sampling methods such as Latin Hypercube Sample [108] or Low Discrepancy Sample [124] may be applied; (*b*) then, from the generated sample, parallel experiments (deployment of application workflows) are run for each parameter set; (*c*) lastly, the surrogate model is trained on the dataset generated in the previous step.

Model Retraining & Application Optimization: once the surrogate model is trained on the sample points previously generated, it is used to explore the optimization search space by deciding the subsequent application configurations to be evaluated in parallel. As soon as the evaluations finish, the model is retrained and optimized asynchronously, then new points are suggested to be evaluated.

Short-time Running Applications

They refer to the case when a few minutes are enough to evaluate a single point in the search space. Such applications also follow the *optimization cycle* previously presented. Besides, they may also use surrogate models to explore the search space. However, differently from *Long-time Running Use Cases*, they can use other optimization techniques such as evolutionary algorithms and swarm intelligence-based algorithms (*e.g.*, Genetic Algorithm [143], Differential Evolution [59], Simulated Annealing [219], Particle Swarm Optimization [73], *etc.*).

3.3.3 Phase III: Finalization

For **reproducibility** purposes, this last phase illustrated at the bottom of Figure 3.3 provides a summary of computations. Therefore, it provides the definition of the optimization problem (optimization variables, objective, and constraints); the sample selection method; the surrogate models or search algorithms with their hyperparameters used to explore the search space of

the optimization problem; and finally, the optimal application configuration found. Providing all this information at the end of computations allows other researchers to verify and reproduce the research results.

Conclusion

This chapter presented rigorous methodologies for, first, **designing reproducible experiments** on the Edge-to-Cloud Continuum, and then, **understanding** and **optimizing the performance** of real-life **Edge-to-Cloud workflows**. Both methodologies support reproducibility and guide users to systematically **define the whole experimental environment** (*i.e.*, computing resources, network, and workflow execution and optimization) and then **share the experiment artifacts** and **results**.
To provide users with a practical exploration of these methodologies across the Edge-to-Cloud Continuum, the next chapter proposes their **implementations as a novel framework, named E2Clab**.

Chapter 4

E2C*lab*: The Methodology Implementation

Contents

This Chapter introduces the E2Clab framework that implements our methodologies. Next, it details the high-level aspects and the architecture of E2Clab and then provides the implementation details. Finally, it discusses how E2Clab can optimize various Edge-to-Cloud applications and support different open scientific testbeds.

4.1 High-Level Aspects and Architecture

To illustrate our methodology for experimentation on the Computing Continuum, we propose E2Clab, a framework that implements it. Researchers may use it to deploy real-life applications on large-scale testbeds and systematically perform meaningful experiments. Please learn more about E2Clab on its documentation Web page [225].

We develop E2Clab with usability in mind: we propose a structure for configuration files (presented in Table 4.1 and discussed in the following sections). Those files allow experimenters to write their requirements descriptively. They are easy to comprehend, use, and adapt to any scenario, reducing the effort of configuring the whole experimental environment.

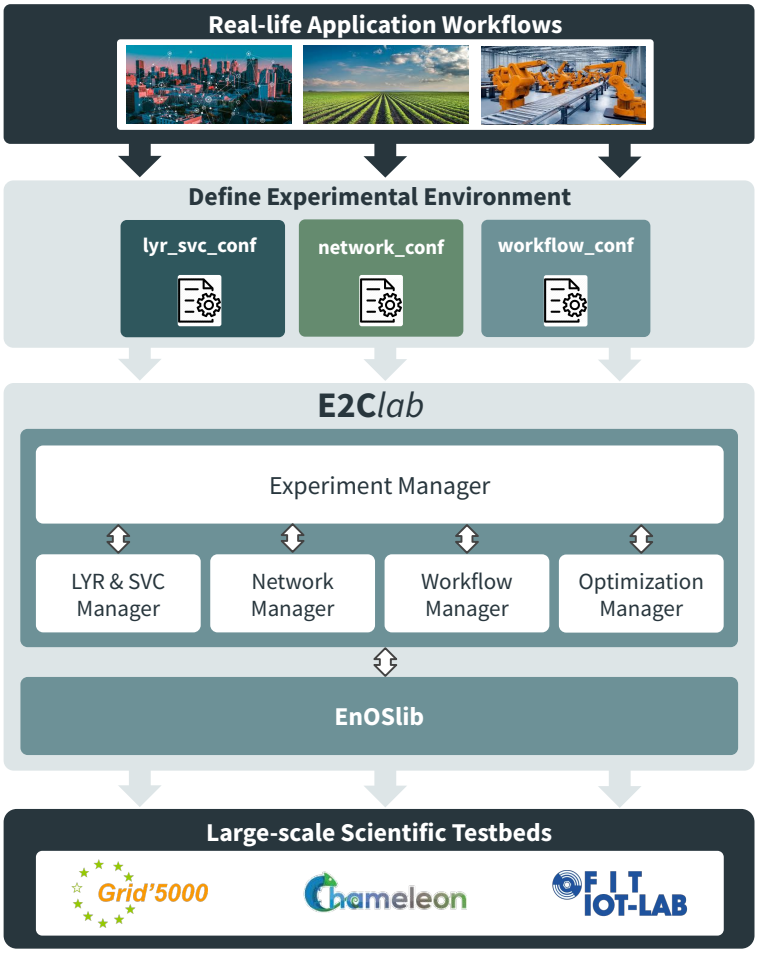


Figure 4.1 – The E2Clab Architecture.

Essentially, using only three configuration files, the experimenter describes the *workflow* (*e.g.*, Edge-to-Cloud data stream processing with Apache Flink), the *layers and services* that compose the scenario (*e.g.*, the Flink master in the Cloud layer and workers on the Edge layer), and the *network* specification (*e.g.*, delays, losses, and rates between the Flink master and workers). E2Clab abstracts from the users the complexity of the mappings between *layers* and *services* with the actual machines in the testbeds (*e.g.*, Grid'5000, Chameleon, FIT IoT LAB).

The E2Clab architecture is depicted at the core of Figure 4.1 and comprises a set of Managers, described in the following sections. Their role is to convert the abstract specifications of *layers* and *services* defined in the configuration file into concrete testbed resources (*e.g.*, physical machines of a computing cluster) with the respective *services* deployed. They also enforce the configuration dependencies of the underlying *services*, defined by experimenters in the *workflow* configuration file.

E2Clab sits on top of EnOSlib [47], a library that brings reusable building blocks for configur-

ing the infrastructure, provisioning software on remote hosts, and organizing the experimental workflow. Interaction with the testbeds is deferred to EnOSlib's providers, and various actions on remote hosts also rely on mechanisms offered by the library (*e.g.*, monitoring stack). E2Clab current version supports Grid'5000, Chameleon, and FIT IoT LAB. However, it can be extended to support other testbeds.

4.2 Implementation

Next, we present the components of the E2Clab framework and their relationships.

4.2.1 Experiment Manager

The *Experiment Manager* coordinates the execution of the whole experiment and the information exchange between all managers. It receives as input all the configuration files provided by the users (*layers and services*, *network*, and *workflow*) and assigns them to the respective managers to set up the experimental environment and execute the experiments. At the end of each experiment, the *Experiment Manager* provides the results to the users.

Table 4.1 presents the structure for the configuration files and a brief description of each of their attributes. These configuration files follow the YAML format [237]. In order to manage the experiment execution logic (*i.e.*, interconnections, life cycle, execution parameters, and execution order) on remote nodes, the *workflow* configuration file follows Ansible's playbook language [10]. Each configuration file has an important role in enabling the 3Rs as they rigorously abstract the underlying infrastructure, allowing other researchers to easily reproduce, replicate or repeat the experiments.

E2Clab provides commands to manage the experiment life cycle. It allows users to divide the experiment deployment into smaller steps, according to the sequence presented in Figure 4.2. For instance, the *e2clab deploy* command automatically enforces all the following commands: *"e2clab* [*layers-services; network; workflow prepare; workflow launch*]*"*; then when the experiment ends (*e.g.*, after a duration defined by the user *"-duration 180"*) E2Clab runs *"e2clab finalize"* (which also includes *"e2clab workflow finalize"*) to backup the relevant data generated, such as application performance metrics; monitoring data; and provenance data captured.

We highlight that one may also easily repeat experiments N times using the *"–repeat N"* argument. Automating the analysis of multiple scenarios (with their respective *.yaml files*), in addition to variations in the workflow configuration parameters, can be easily achieved using the *"–scenarios_name"* and *"–app_conf"* arguments, respectively. Furthermore, the *workflow* deployment step can also be divided as *"e2clab workflow* [*prepare, launch, or finalize*]*"*. Finally, the *"e2clab optimize"* command helps users to find the optimal application workflow configurations. It automatically enforces all the commands executed by *"e2clab deploy"*.

Table 4.1 – E2Clab Configuration Files

	Attributes	**Description**
Layers & Services	environment g5k chameleon iotlab chiedge	Refers to the testbeds to run the experiments on, such as Grid'5000, Chameleon Cloud, FIT IoT LAB, or Chameleon Edge.
	monitoring name server	Deploys a monitoring service. It may be a TIG stack (Telegraf + InfluxDB + Grafana) or a TPG stack (Telegraf + Promotheus + Grafana).
	provenance name server	Deploys a provenance data capture service. It allows capturing provenance data in IoT/Edge devices and Cloud/HPC systems.
	layers - name	Defines the hierarchy between services and group them in different granularities.
	services - name environment roles quantity repeat	Defines service-specific configurations, such as the testbed it should be deployed on if it should be monitored (*roles:* [*monitoring*]), the number of nodes required to deploy the service, the service repeatability, among others.
Network	networks - src dst delay rate loss	Defines custom network configurations between the layers. For each network, users may vary parameters such as delay, rate, and loss.
Workflow	- hosts	Refers to remote nodes (hosting the defined services) to deploy the experiment artifacts (libraries, applications, dataset, among others) and to enforce the experiment workflow.
	depends_on - service_selector grouping prefix	Represents dependencies between services that compose a workflow. These services may be grouped using different strategies. The goal is to access from a service(s) metadata (such as IP address, port number, configurations, among others) of another service(s).
	prepare launch finalize	Refers to the three phases of the workflow management. Each one consists of tasks to be applied on remote nodes such as 1) prepare: copy files to remote nodes (libraries, dataset, software, *etc.*); 2) launch: execute commands on nodes to start applications with specific parameters or resource constraints; 3) finalize: backup data generated during execution of experiments.

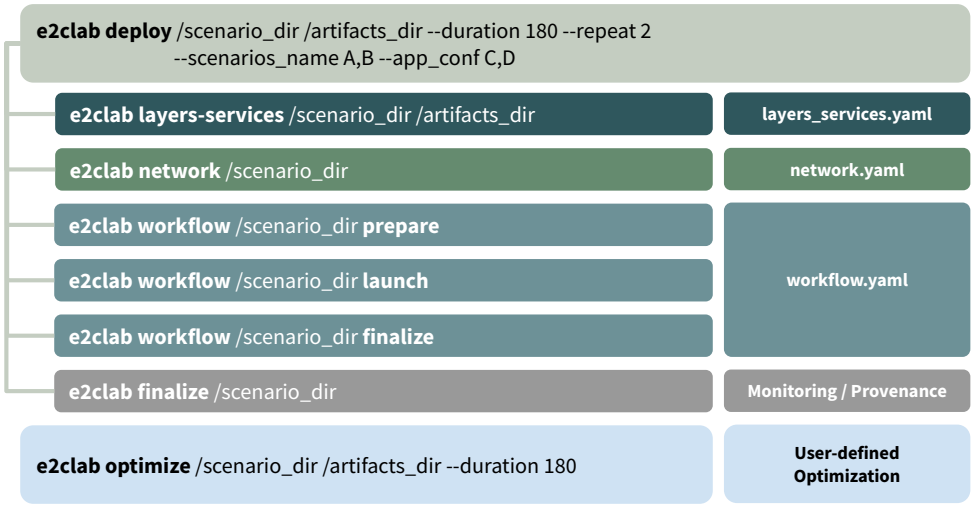


Figure 4.2 – E2Clab CLI to manage the experiment life cycle.

4.2.2 Layers and Services Manager

This *Manager* interprets the *layers and services* configuration file and starts the deployment phase by reserving Edge-to-Cloud physical resources on multiple testbed environments (*e.g.*, Grid'5000, Chameleon, FIT IoT LAB) and then installing, configuring, and deploying all the *services* defined by the user. It also allows users to deploy additional services, such as a *Monitoring Service* (monitors resource usage) and a *Provenance Data Capture Service* (captures application workflow data). Both services provide visualization tools to track workflow execution.

Deploying Edge-to-Cloud workflows with E2Clab. Deploying applications using E2Clab requires their implementation as a *User-Defined Service*. Users services must inherit the *Service* class provided by E2Clab. Users have to implement the logic of their service (according to their needs) in the *deploy*() method and then register it using the *register_service*() method.

Publicly accessible in our repository, users may find and reuse in their experiments several services already implemented. The available services refer to master-worker-based services, containerized services using Docker, and services composed of sub-services. To cite a few examples: Horovod [195] service (for distributed Deep Learning training); Flower [30] service (for Federated Learning training); Apache Flink [44] service (for distributed data stream processing); and COMPSs [21] service (for executing parallel applications).

Illustrative example: Edge-to-Cloud deployment of COMPSs workflows on Chameleon Cloud and CHI@Edge testbeds. COMPSs is a framework that aims to ease the development and execution of parallel applications for distributed infrastructures. Listing 4.1 provides an excerpt of a configuration file where the user aims to deploy a COMPSs cluster on Chameleon Cloud

```yaml
environment:
  chameleon:
    rc_file: app-cred-cloud-openrc.sh
    key_name: my-sshkey
    image: CC-Ubuntu20.04
  chiedge:
    rc_file: app-cred-edge-openrc.sh
monitoring:
  name: tig
  cluster: compute_skylake
layers:
- name: cloud
  services:
  - name: compssMaster
    environment: "chameleon"
    cluster: gpu_rtx_6000, quantity: 1, roles: [monitoring]
- name: edge
  services:
  - name: compssWorker
    environment: "chiedge"
    cluster: jetson-nano, quantity: 3, roles: [monitoring]
```

Listing 4.1 – Layers and Services configuration file example.

and CHI@Edge testbeds. Line 1 refers to the testbeds to deploy the COMPSs cluster, and line 8 defines the monitoring service.

Next, the user defines the layers and services. Line 14 refers to the *compssMaster* service on the *cloud* layer defined in line 12. Then, the *compssWorker* service (line 19) composed of 3 Edge devices ($quantity$: 3) are deployed on the *Edge* layer (see line 17). Both services will be monitored during execution ($roles$: [$monitoring$]). It means that users can monitor the computing resources of the COMPSs cluster during the execution of experiments.

Suppose one needs to deploy two (or more) distinct regions of COMPSs workers on the Edge layer while keeping the same configuration on the Cloud side (COMPSs master). In that case, it can be done by simply adding the attribute ($repeat$: 2) within the *compssWorker* service. This allows users to quickly and transparently scale the scenarios by adding *layers* and to easily vary them by replicating the *services*.

Listings 4.2 and 4.3 present the user-defined service implementation of the *COMPSs Master* and *COMPSs Worker* services. In this example, the COMPSs cluster is deployed using Docker containers, with images obtained by default from the Docker Hub [71]. Hence, users can set their custom images by using the $image$ attribute and specifying the image identifier in the Docker Hub (*e.g., image: "username/my-compss-version"*).

We highlight that leveraging containerization and virtualization enables **reproducibility**: everything required to run the *service* efficiently and bug-free (*i.e.,* configuration files, libraries, dependencies, datasets, *etc.*) can be packed, made publicly available and reused. Figure 4.3 illustrates the deployment configuration defined in Listing 4.1.

```python
from e2clab.services import Service
import enoslib as en

class CompssMaster(Service):
    def deploy(self):
        self.deploy_docker()
        # Start COMPSs Master container
        with en.actions(roles=self.roles) as a:
            a.docker_container(name="compss_master", image="compss/compss")
        # Register the Service
        return self.register_service()
```

Listing 4.2 – COMPSs master service example.

```python
from e2clab.services import Service
import enoslib as en

class CompssWorker(Service):
    def deploy(self):
        self.deploy_docker()
        # Start COMPSs Worker containers
        workers = []
        extra_compss_worker = []
        for host in self.hosts:
            worker_id = f'{compss_worker}_{host.alias}'
            workers.append(worker_id)
            extra_compss_worker.append({'container_name': worker_id})
            with en.actions(roles=host) as a:
                a.docker_container(name=worker_id, image="compss/compss",
                                   published_ports="43001-43002:43001-43002")
        # Register the Service
        return self.register_service(extra=extra_compss_worker)
```

Listing 4.3 – COMPSs worker service example.

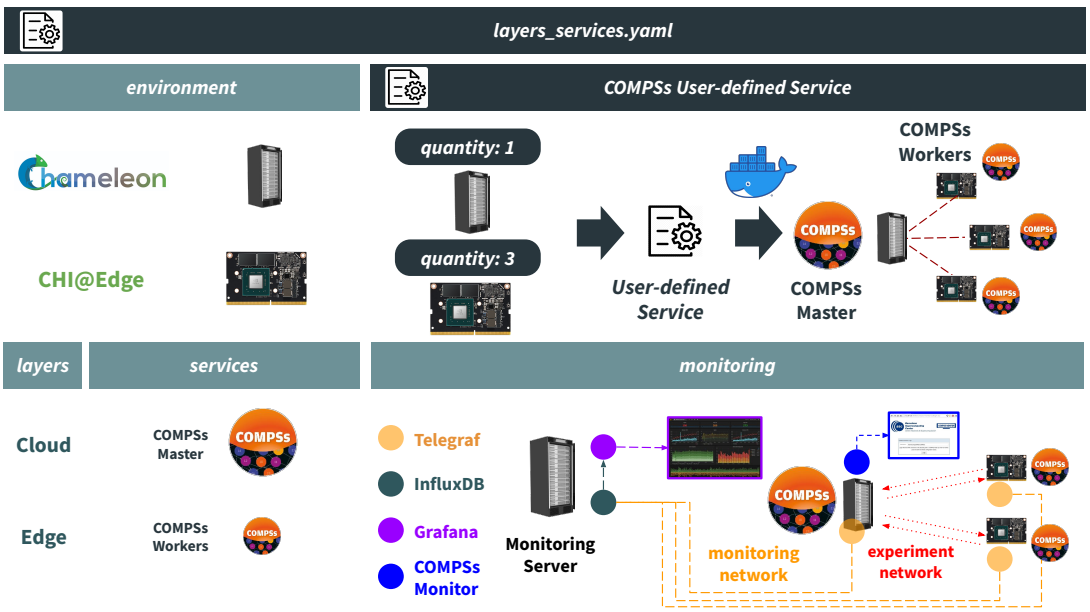


Figure 4.3 – Deployment of a COMPSs cluster on Chameleon Cloud and CHI@Edge.

```
1  networks:
2  - src: cloud
3    dst: edge
4    delay: 50ms
5    rate: 1gbit
6    loss: 2
7  - src: cloud.1.compss.master.1
8    dst: edge.1.compss.worker.2
9    delay: 280ms
10   rate: 150mbit
11   loss: 2
```

Listing 4.4 – Network configuration file example.

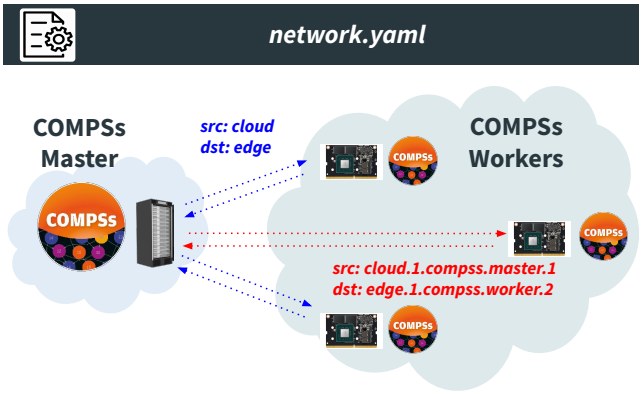


Figure 4.4 – E2Clab Network: communication constraints for the COMPSs cluster.

4.2.3 Network Manager

After the former *Manager* leases the testbed resources and deploys the services, the *Network Manager* interprets the *network.yaml* (see Listing 4.4) file to enforce the network configurations defined by users (*e.g.*, delay, loss, and bandwidth). This Manager defines the communication rules between *layers* (coarse-grained approach), or in a finer granular way, between *layers and services* or between *services*. It uses NetEm [131] (Linux traffic control) to configure the whole network, and once this is done, it generates a file report for users to check and validate the network parameters.

Regarding the COMPSs cluster example, the *network.yaml* file defines two distinct networks rules, as shown in Listing 4.4 lines 2 and 7. The first rule (coarse-grained) is applied between the *Cloud* and *Edge* layers, which means between the *master* and *all workers*. The second one (fine-grained) is applied between the *master* and *the second worker*. The second rule enforces a delay of 280ms, 150mbit bandwidth, and a packet loss of 2%. Figure 4.4 illustrates the network configurations defined in Listing 4.4.

We highlight that E2Clab abstracts all the complexities of configuring the whole network from users. E2Clab allows users to enforce complex rules by using high-level names, such as "$src : cloud$" and "$dst : edge$"; or "$src : cloud.1.compss.master.1$" and "$dst : edge.1.compss.worker.*$" (the "$*$" symbol may be used to refer to all workers). Such abstraction of the network configuration complexity from end users enables the E2Clab **replicability** and **reproducibility**.

4.2.4 Workflow Manager

This *Manager* interprets the *workflow.yaml* configuration file to manage and automate the life cycle of each *service* through Ansible tasks. It allows users to deploy their applications in three well-defined steps: *prepare*, *launch*, and *finalize*. The *Workflow Manager* defines the execution

```
1  - hosts: cloud.compss.1.master.1
2    depends_on:
3      service_selector: "edge.compss.1.worker.*"
4      grouping: "aggregate"
5      prefix: "workers"
6    prepare:
7      # Generating the project.xml file on the master.
8      - copy:
9          src: gen_project_xml.py
10         dest: /tmp/gen_project_xml.py
11     - shell: python gen_project_xml.py --workers {{ workers.container_name }}
12   launch:
13     # Running a COMPSs application
14     - shell: docker exec -it compssmaster bash -c runcompss
15                --project="project.xml" --resources="resources.xml" -d simple.py
16   finalize:
17     # Backup processing latency logs
18     - fetch:
19         src: /opt/metrics/processing-latency
20         dest: /experiment-results/master/metrics/
```

Listing 4.5 – E2Clab Workflow: managing services in the COMPSs cluster.

order and relationships between services. Considering the COMPSs cluster example, Listing 4.5 presents how E2Clab avoids burdening the users with doing the exact mapping between *layers* (Cloud and Edge) and *services* (Master and Workers), at the *workflow* level.

Line 1 refers to the management tasks to be applied on the COMPSs master. The *prepare* phase (see line 6) is composed of two tasks: first, it copies the Python script to generate the *project.xml* file (defines the resources used by the master); and then generates it. We highlight that using the "$depends\_on$" attribute (see line 2), the *master* can access data from other services, in this case, data from all workers ("$service\_selector$: $edge.compss.1.worker.*$") are aggregated ($grouping$: $aggregate$). Then, by using the prefix $workers$ the master can get the workers ids (container name) and pass them as an argument to the Python script that generates the *project.xml* file (*e.g.,* "–*workers* {{*workers.container_name*}}"). Accessing this metadata is possible since users saved it during the service registration (see Listing 4.3 line 18). After that, the *launch* phase (see line 12) executes the application on the COMPSs master container.

Finally, when the application execution ends the *finalize* phase (see line 16) backups (*e.g.,* fetches from the remote to the local node) log files that the user wants to analyze. We highlight that the *Workflow Manager* collects all data generated during the execution of the experiments, such as physical machine monitoring, files generated by user applications such as log files, files containing performance metrics data, *etc.* Researchers must provide all the data collected by the Manager in a public and safe repository for **replicability** and **reproducibility** purposes.

4.2.5 Optimization Manager

The *Optimization Manager* implements the optimization approach in Figure 3.3. Its role is to: interpret the user-defined optimization setup defined by users and then automate the *op-*

timization cycle (1. parallel deployment of the application workflow in a large-scale testbed; 2. simultaneous application workflow execution; 3. asynchronous model training and optimization; and 4. reconfiguration of the application workflow for a new evaluation) to optimize the application workflow. Lastly, the *Optimization Manager* provides a summary of computations for reproducibility purposes.

The *Optimization Manager* takes advantage of *Ray* [170] to run parallel application workflows on large-scale testbeds (*e.g.*, Grid'5000, Chameleon, FIT IoT LAB). Ray Tune [130] provides state-of-the-art search algorithms, manages model checkpoints and logging, and methods for analyzing training.

Optimizing Edge-to-Cloud workflows with E2Clab. Optimizing application workflows using E2Clab requires the implementation of the optimization logic as a *User-defined Optimization*, see an example in Listing 4.6. The *Optimization Manager* offers a class-based API that allows researchers to set up and control the optimization.

Users have to implement two functions: *run*() and *run_objective*(). First, users have to inherit the *Optimization class* and define in the *run*() function the objective (*e.g.*, minimize the application processing time), the optimization variables, and their constraints (*e.g.*, the number of processing cores to be assigned for the various tasks that compose the application), search algorithms (*e.g.*, single-objective and multi-objective Bayesian Optimization search algorithms from libraries such as *Scikit-Optimize* [194], *Dragonfly* [116], *Ax* [23], *HEBO* [56], among others), parallelism (*e.g.*, N parallel evaluations of the application workflow), among others.

Next, users define in the *run_objective*() function (Listing 4.6 line 28) their optimization logic, which runs in parallel to train the model. To do so, the *Optimization class* provides the following three methods:

1. *prepare*(): for reproducibility of optimization evaluations, it generates a dedicated optimization directory for each model evaluation (Listing 4.6 line 30).

2. *launch*(): deploys the application on large-scale testbeds to perform a model evaluation (Listing 4.6 line 32). For reproducibility, deployment-related information is captured, such as physical machines, network constraints, and application configurations.

3. *finalize*(): for reproducibility purposes, it stores the optimization computations for a given model evaluation in the optimization directory created in the *prepare*() phase (Listing 4.6 line 34). Saved information refers to intermediate models throughout training and points evaluated.

```python
from e2clab.optimizer import Optimization

class UserDefinedOptimization(Optimization):

    def run(self):
        algo = SkOptSearch(
                    optimizer = Optimizer(
                        base_estimator='ET',
                        n_initial_points=45,
                        initial_point_generator="lhs",
                        acq_func="gp_hedge"))
        algo = ConcurrencyLimiter(algo, max_concurrent=2)
        scheduler = AsyncHyperBandScheduler()
        objective = tune.run(
            self.run_objective,
            metric="processing_time",
            mode="min",
            name="compss_application",
            search_alg=algo,
            scheduler=scheduler,
            num_samples=10,
            config={
                "task_1": tune.randint(20, 60),
                "task_2": tune.randint(20, 60),
                "task_3": tune.randint(20, 60),
                "task_4": tune.randint(3, 9)})

    def run_objective(self, _config):
        # create an optimization directory
        self.prepare()
        # deploy the configs on the testbed
        self.launch()
        # backup the optimization computations
        self.finalize()
        # report the metric value to Ray Tune
        tune.report(user_resp_time=user_resp_time)
```

Listing 4.6 – Example of a user-defined optimization in **E2C***lab*.

4.3 Discussion

Besides revealing the hidden performance trade-offs of workflow deployments through reproducible experiments, E2Clab exhibits a series of features that make it a promising platform for future performance optimization of applications on the Edge-to-Cloud Continuum. We briefly discuss them here.

4.3.1 Usability and Reusability

E2Clab targets **usability** by abstracting all the low-level details of the definition and configuration of the experimental environment. It avoids the burden of mapping *layers* and *services* to users since it provides a high-level abstraction that allows performing this mapping through

logical names and pattern matching. Besides, the configuration files are designed to be easy to use and understand.

Furthermore, it targets **reusability** by formalizing the support of users applications through *User-Defined Services*. For this purpose, E2Clab provides a *Service* class in which users have to override a *deploy* method to define the deployment logic of their services (*e.g.,* mapping the services to the physical machines; installing required software; *etc.*) and then register them. Next, E2Clab managers can deploy each service on the different testbeds. In our repository, users may find and reuse in their experiments several services already implemented, such as Horovod [195]; Flower [30]; Apache Flink [44]; COMPSs [21]; among others.

4.3.2 Methodology Genericness

Our experiment methodology is generic: E2Clab provides two simple abstractions for modeling applications and infrastructures: *layers* and *services*. Such abstractions are powerful enough to express several applications (through *User-defined Services*) deployed on heterogeneous testbed environments, ranging from the Edge to the Cloud (*e.g.,* Grid'5000, Chameleon, and FIT IoT LAB). Furthermore, we believe that the core idea behind E2Clab, of a methodology to enable the design of relevant testbeds for 3R's experiments, may prove useful for understanding and optimizing the performance of large-scale applications.

Supporting different Edge and Cloud testbeds: the deployment of Edge-to-Cloud workflows on multiple testbeds is supported by our experiment methodology. It allows users to analyze application workflows on various large-scale scientific testbeds, such as Grid'5000 and Chameleon (provide Cloud/HPC resources) and FIT IoT LAB and CHI@Edge (provide IoT/Edge devices). We highlight that the definition of the experimental environment through E2Clab configuration files (*e.g.,* layer_services.yaml, network.yaml, and workflow.yaml) is agnostic from the testbed, meaning that a deployment done in the Grid'5000 testbed may be replicated in Chameleon if the later also provides the required computing resources for the deployment.

Our optimization methodology is generic: the optimization of other applications may be achieved by describing the application optimization problem in the *User-defined Optimization* file. We highlight that despite our evaluations focusing on the Pl@ntNet as a use case (presented in the next chapter), our methodology and its implementation in E2Clab can be used to analyze other applications in the context of the Edge-to-Cloud Continuum. Furthermore, it allows users to define different optimization problems (*e.g.,* single-objective and multi-objective problems) with application-specific optimization variables and constraints.

Conclusion

This chapter proposed the E2Clab framework that implements our methodologies. E2Clab aims to overcome the **complexities of understanding and optimizing** Edge-to-Cloud workflows. It helps users to execute **reproducible experiments** on open scientific testbeds providing access to heterogeneous Edge-to-Cloud computing resources.
E2Clab supports the **complete analysis cycle** of an application on the Computing Continuum: **experiment design** (*i.e.,* defining the computing resources, network constraints, and workflow execution), **deployment** (e.g., on Grid'5000, Chameleon, CHI@Edge, and FIT IoT LAB), **execution** and **monitoring**, and application **optimization**. Furthermore, **E2Clab is generic** regarding application deployment, supported testbeds, and application optimization.
To illustrate the benefits of exploring E2Clab, the next chapter presents a validation with Pl@ntNet, a global plant identification application.

Chapter 5

Experimental Evaluation and Validation with the Pl@ntNet Application

Contents

The E2Clab framework aims to enable the Computing Continuum vision by allowing researchers to perform reproducible experiments on large-scale testbeds in order to understand and optimize the performance of Edge-to-Cloud workflows.

Toward this goal, this Chapter validates the E2Clab framework with a real-life application. It shows how E2Clab helps to understand and optimize the performance of Pl@ntNet and supports reproducible experiments.

5.1 Pl@ntNet: A Real-life Botanical Observation Application

Pl@ntNet is a participatory application and platform that produces botanical data and plant identification. The data comprises +35K plant species collected from more than 200 countries. As illustrated in Figure 5.1, using the Pl@ntNet mobile application, users (located in the Edge) may identify plants from pictures taken by their phones. Before sending these pictures, some preprocessing is done to reduce the image size.

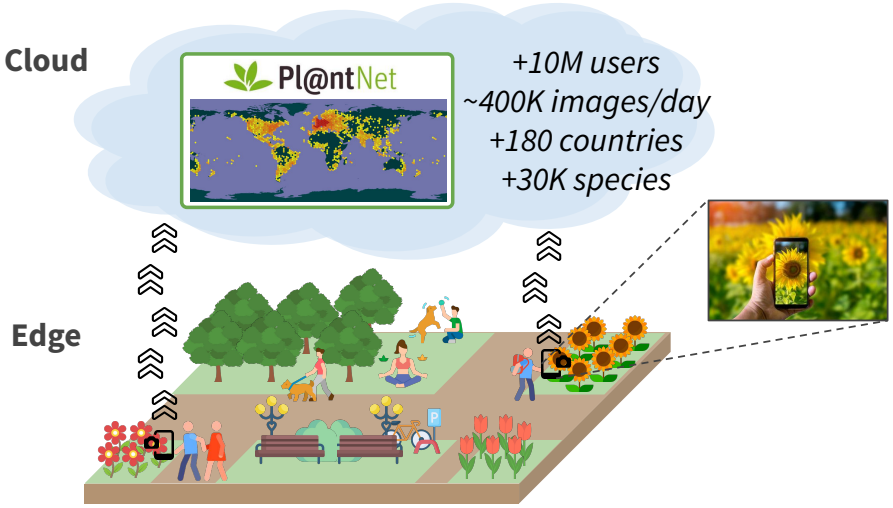


Figure 5.1 – The Pl@ntNet application.

Then, the Pl@ntNet **Identification Engine** (located in the Cloud), subject to analysis in this work, is responsible for automatically identifying species through Deep Learning. In a nutshell, the Identification Engine performs two main activities: (1) *Species prediction:* refers to the feature extraction and classification of user images; and (2) *Similarity Search:* searches for the images of the botanical databases that are the most similar to the user images. At the end of the processing, the Identification Engine returns the ranked list of most probable species with their respective, most similar plant pictures, allowing interactive validation by the users.

The processing performance of the Identification Engine strongly depends on the **thread pool size** configured to process the various **tasks** involved during the identification of users images. Table 5.1 presents the execution order of all tasks, the thread pool they belong to, and in which hardware they take place. Table 5.2 describes the role of each thread pool and an example of the configuration currently used in the Pl@ntNet production servers. This configuration was defined by Pl@ntNet engineers based on their best practical experience with the Pl@ntNet system considering mainly the following: (a) for thread pools using CPU: a machine with 40 CPU cores available; and (b) for the GPU thread pool: the maximum number of threads which fit in GPU memory.

The main performance metric for this application is the **user response time**. A preliminary analysis [15] showed that to achieve a 4 seconds response time (the maximum tolerated by users), the thread pool and hardware configurations can not serve more than 120 simultaneous requests (3.86 ± 0.13), as shown in Figure 5.2.

In this context, meaningful questions are: *Is there a better thread pool allocation that minimizes the user response time? How many more users can the system serve if we find a better thread pool configuration?* The answers to those questions and more analytical insights will be presented in Section 5.2

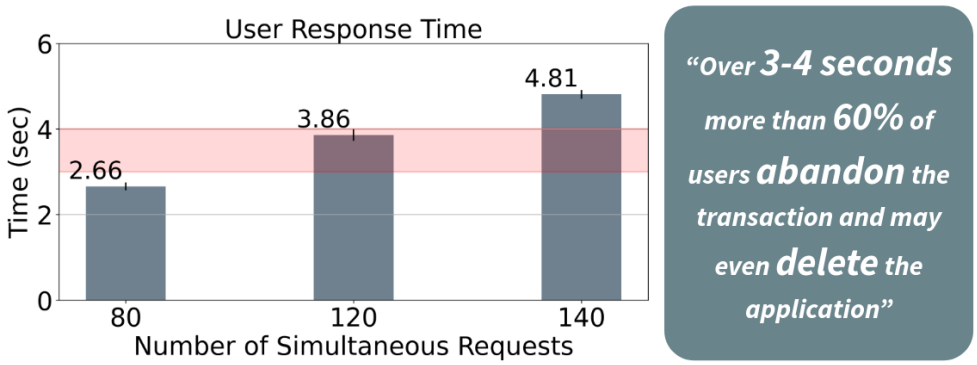


Figure 5.2 – Pl@ntNet Engine: user response time.

through our proposed methodologies and their implementation in the E2Clab framework.

Let us highlight that Pl@ntNet is representative of other applications in the context of the Computing Continuum. As illustrated in Figure 5.1, it consists of many geographically distributed devices (over 10 million users) that collect and send data (about 400K plant images per day), and perform preprocessing at the Edge, followed by extensive processing (*e.g.*, species prediction, similarity search, *etc.*) in centralized Cloud/HPC infrastructures.

5.2 Research Questions and Experimental Setup

In this section, we illustrate our proposed methodologies by showing how they can be used to analyze the performance of the Pl@ntNet botanical application and to find its thread pool configurations.

The goal of our experiments is to answer the following research questions:

1. What software configuration minimizes the user response time?
2. How does the number of simultaneous users accessing the system impact the user response time?
3. How do the *Extraction* and *Similarity Search* thread pool configurations impact the processing time and user response time?

The experimental setup is defined as follows:

a) Scenario Configuration: the experiments are carried out on 42 nodes of the Grid'5000 [32] testbed (clusters [92]: *chifflot*, *chiclet*, *chetemi*, *chifflet*, and *gros*). Since the *Pl@ntNet Identification Engine* requires GPU, it is deployed on the *chifflot* machines (model Dell PowerEdge R740), which are equipped with Nvidia Tesla V100-PCIE-32GB GPUs, Intel Xeon Gold 6126 (Skylake, 2.60GHz, 2 CPUs/node, 12 cores/CPU), 192GB of memory, 480GB SSD, and 25Gbps Ethernet

Table 5.1 – Identification processing steps.

Task	Description	Thread pool	Hardware
pre-process	Decoding the query parameters.	HTTP	CPU
wait-download	Wait for an available download thread.	HTTP, Download	CPU
download	Download images.	Download	CPU
wait-extract	Wait for an available extractor thread.	HTTP, Extract	CPU, **GPU**
extract	DNN inference of the image.	Extract	**GPU**
process	Process classification and similarity search output at query level.	HTTP	CPU
wait-simsearch	Wait for an available similarity search thread.	HTTP, Simsearch	CPU
simsearch	Search the most similar images in our database.	Simsearch	CPU
post-process	Check processed query results and format the response.	HTTP	CPU

Table 5.2 – Thread pool configuration of Pl@ntNet Engine.

Thread pool	Size (# threads)	Description	Hardware
HTTP	40	# simultaneous requests being processed.	CPU
Download	40	# simultaneous images being downloaded.	CPU
Extract	7	# simultaneous inferences in a single GPU.	**GPU**
Simsearch	40	# simultaneous similarity search.	CPU

interface. The clients submitting requests to the *Pl@ntNet Identification Engine* are deployed on the *chiclet*, *chetemi*, *chifflet*, and *gros* clusters. The network connection is configured with 10Gb.

b) Workloads: we defined three categories of workloads, according to the number of simultaneous requests (*i.e.*, 80, 120, and 140) submitted to the *Pl@ntNet Identification Engine* during the whole experiment execution.

c) Configuration Parameters: Table 5.2 presents the parameters used to configure the thread pool size of the *Pl@ntNet Engine*. As presented in Equation 5.1, these parameters refer to the optimization variables of the optimization problem.

d) Performance Metrics: the metric of interest is the *user response time*. In Equation 5.1, this metric is to be minimized as the optimization objective. The *user response time* refers to the average time that a user waits for the response to a request. Besides this metric, we also analyze the *identification processing time*, which refers to the average time to process a user request. The identification processing is divided into multiple tasks running in parallel, as described in Table 5.1.

We compare and analyze the *user response time* and *identification processing time* with respect to two thread pool configurations: ***baseline*** and ***preliminary optimum***. The ***baseline*** refers to the current *Pl@ntNet* configuration used in the production servers. This configuration was defined by *Pl@ntNet* engineers based on their best practical experience with the *Pl@ntNet* system, as explained in Section 5.1 and presented in Table 5.2.

The preliminary optimum configuration: this configuration is the one found using our methodology (see Subsection 5.3.1). We named it preliminary since the optimization problem may have multiple *minima* and one may find other application configurations if a different technique is used (*e.g.*, Gaussian Process (Kriging) [203], Gradient Boosting Regression Trees [89], among others).

Besides, changes in the hardware configuration (*e.g.*, size of GPU memory, number of CPU cores, among others) running the Pl@ntNet application will require a new search for the thread pool sizes since their configuration strongly depends on the hardware. In this case, our optimization methodology should be applied again. In a subsequent step, we further refine the preliminary optimum using sensitivity analysis to obtain what we call ***refined optimum*** (see Subsection 5.3.3).

Since we identified variations between measurements through experiments, we decided to repeat each experiment (each thread pool configuration) 7x to obtain accurate measurements. Besides, we ran each experiment for 23 minutes (or 1380 seconds) with an interval of metric collection of 10 seconds to minimize the standard deviation of the metrics collected. We use 23 minutes because this amount of time is enough to stress the Pl@ntNet Identification Engine and obtain more stable measurements of the user response times. Therefore, the *user response time* is presented with the mean and standard deviation regarding 966 measurements ($138 * 7$). Furthermore, thanks to the **repeatability** feature provided in E2Clab, one may repeat those experiments easily by issuing the following command:

```
e2clab deploy –repeat 7 –duration 1380 experiments/ artifacts/
```

5.3 Evaluation

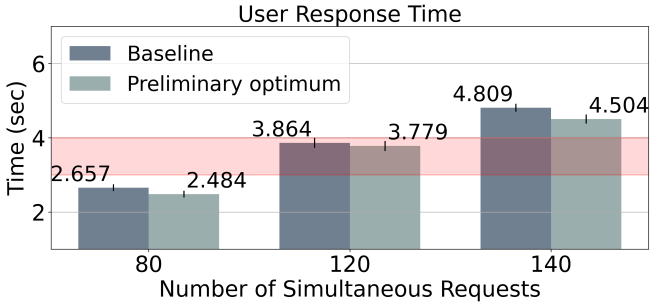
Next, we present the results of the three research questions.

5.3.1 What software configuration minimizes the user response time?

The optimization problem to be solved can be stated as follows:

Figure 5.3 – (left) Baseline vs. preliminary optimum configurations, and (right) User response time: baseline vs. preliminary.

Thread pool	baseline	preliminary optimum
HTTP	40	54
Download	40	54
Extract	7	7
Simsearch	40	53
User response time	**2.657** (± 0.0914)	**2.484** (± 0.0912)



$$
\begin{array}{lll}
\text{Find} & (http, download, simsearch, extract), \ in\ order\ to & \\
\text{Minimize} & UserResponseTime & \\
\text{Subject to} & 20 \leq (http, download, simsearch) \leq 60, \ Pool\ Size. & \\
 & 3 \leq (extract) \leq 9, \qquad\qquad\qquad\quad Pool\ Size. &
\end{array}
\tag{5.1}
$$

The function *UserResponseTime* is given by the parallel execution of the Pl@ntNet workflow on the Grid'5000 testbed, as described in *Phase II* of our optimization methodology.

In order to define the search space dimensions, we run experiments to identify the maximum upper bounds of variables that do not increase the *user response time* compared to the baseline Pl@ntNet configuration. Therefore, the lower and upper bounds of variables (see Equation 5.1) are $\pm 50\%$ of the baseline configuration (recall Table 5.2), respectively.

The workload uses 80 simultaneous requests to the Pl@ntNet Identification engine. We highlight that this number has to be bigger than the upper bound of the *HTTP thread pool size* since the *HTTP pool* refers to the simultaneous requests being processed.

We leverage Bayesian Optimization since it is typically used for global optimization of black-box functions that are expensive to evaluate [88]. *Extra Trees* regressor is used as surrogate model [194] to model our expensive function. This surrogate model is improved by evaluating the *UserResponseTime* function at the next points. The goal is to find the minimum of *UserResponseTime* function with as few evaluations as possible. After nine evaluations, the minimization has converged, and the results are presented in Figure 5.3 (left), considering a workload of 80 simultaneous requests. As one may note, the preliminary optimum configuration reduces the user response time by 7% and can serve 35% more simultaneous users (54 against 40, see the *HTTP* thread pool).

From the results, we highlight that thanks to our methodologies implemented in E2Clab, one may easily find an optimized application configuration. E2Clab abstracts all the complexities to: define the whole optimization problem; deploy the application on large-scale scientific testbeds;

run parallel evaluations of the optimization; and collect all the experiments results.

In the following sections, we enhance our analysis to (a) understand the performance of both configurations for different workloads; and (b) better understand the performance results and their correlation with resource usage.

5.3.2 How does the number of simultaneous users accessing the system impact the user response time?

In order to understand the impact of different workloads on the *user response time*, we defined three workloads that represent simultaneous requests submitted to the Pl@ntNet system. These experiments aim to compare the performance gains of the preliminary optimum thread pool configuration (found using our methodology) against the baseline (current Pl@ntNet configuration). Lastly, we exploit the maximum number of simultaneous requests that each configuration can handle considering the constraint of 3-4 seconds *user response time*.

As presented in Figure 5.3 (right), we scale up the workloads as follows: 80, 120, and 140 simultaneous requests. As one may note, the preliminary optimum configuration outperforms the baseline for all workloads. We highlight that the difference between them varied as follows: 6.9%, 2.2%, and 6.7% for 80, 120, and 140 simultaneous requests, respectively.

The main observation is that the preliminary optimum configuration (found using our methodology) outperforms the baseline thanks to a better thread pool allocation that allows the Pl@ntNet system to serve simultaneously 35% more requests (54 against 40) with a shorter user response time when compared to the baseline.

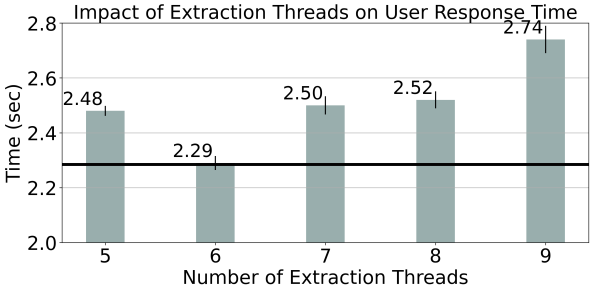
We also highlight that thanks to the transparent scaling feature provided by E2Clab, one may easily scale up the workloads to analyze their impact on the application performance.

5.3.3 How do the *Extraction* and *Similarity Search* thread pool configurations impact the processing and user response times?

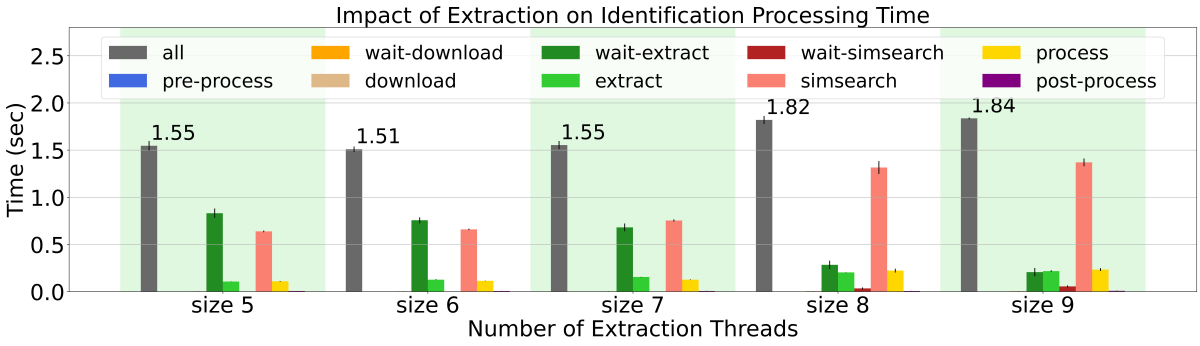
Since the *extraction* and *similarity search* tasks are the most time-consuming compared to the remaining ones, we zoom our analysis on them. The goal is to improve the thread pool configuration even more and identify possible bottlenecks on the Pl@ntNet identification engine.

The experiment aims to understand how variations in the preliminary optimum thread pool configuration of the *extraction* and *similarity search* tasks impact the user response time and the processing time of the identification tasks.

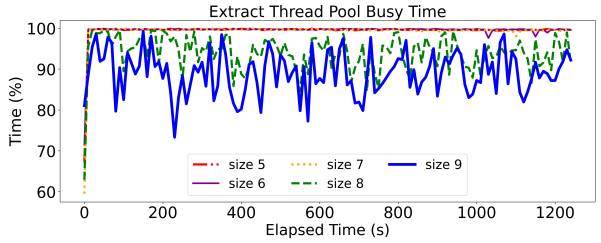
We apply *Sensitivity Analysis* techniques to explore the impact of such variations. From the existing Sensitivity Analysis methods, we decided to use *One-at-a-time* (*OAT*) [104]. OAT is a simple and common approach consisting of varying a single parameter at a time to identify the effect on the output.



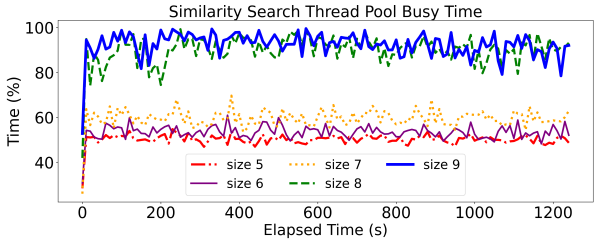
(a) user response time.



(b) processing time.



(c) extract pool busy time.



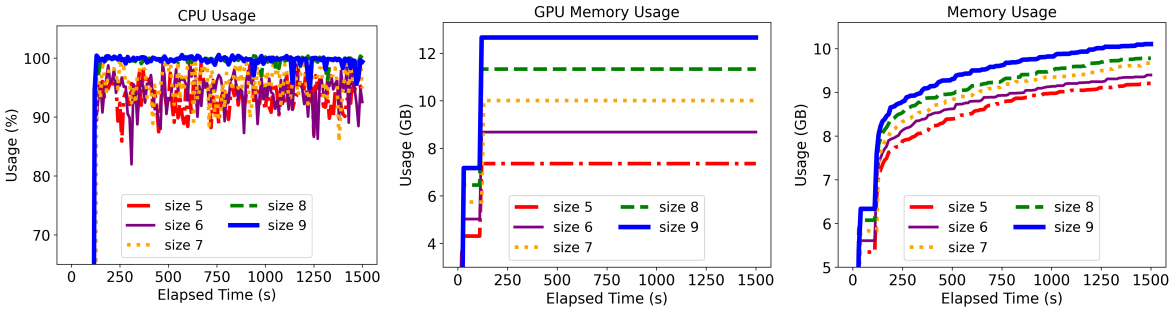
(d) simsearch pool busy time.

Figure 5.4 – Impact of extract thread variability.

In our case, the parameters are *extract* and *simsearch* thread pool sizes. We vary the *extract* pool size in ± 2 from the current size (7 threads), while the *simsearch* in ± 3 (current size is 53 and for simplification, we do not present in Figure 5.6a the times for 50 and 51 since they are bigger than 52). These variations result in 10 new thread pool configurations to be evaluated. Therefore, we take advantage of E2Clab to automatically run them in a reproducible way, following E2Clab's methodology.

Analyzing thread pool variations in the extraction task

Figure 5.4 shows the impact of extraction threads on (a) the user response time, (b) the time to process each task, (c) extract pool busy time, and (d) simsearch pool busy time. Furthermore, presented in Figure 5.5, we also analyze their impact on resource usage, such as (a) CPU usage



(a) CPU usage. (b) GPU memory usage. (c) system memory usage.

Figure 5.5 – Impact of extract thread variability on resource consumption.

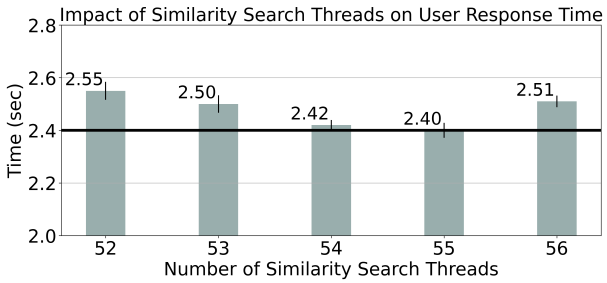
(*b*) GPU memory, (*c*) system memory.

In Figure 5.4a, we observe that the preliminary optimum configuration with seven extract threads does not produce the minimum user response time since using six extract threads reduces it by 8.5%. Decreasing to 5 threads or increasing it to 8 or 9 threads impacts negatively when compared to 6 threads. The explanation for this behavior is given next.

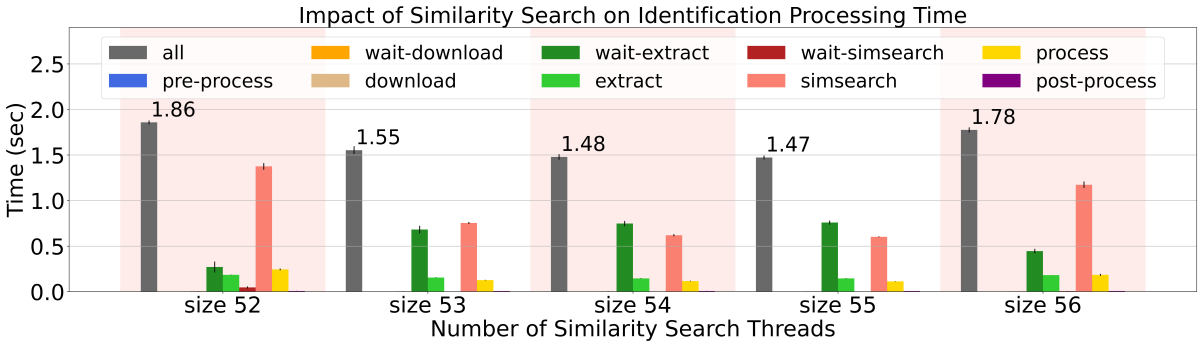
Regarding the processing time (Figure 5.4b), as expected, the *wait-extract* time reduces as we increase the number of *extract* threads, while the *simsearch* task time increases. This time increase in the *simsearch* task can be explained by Figure 5.5a since using 8 and 9 extract tasks results in a CPU usage of 100% during the whole application execution. Hence, as those tasks compete for processing resources, allocating more *extract* threads impact negatively the *simsearch* task time. As for the remaining sizes, they varied between 85% and 100%. This behavior explains the results observed for the user response time in Figure 5.4a. Furthermore, unlike the *wait-extract* time, the *extract* task time was not reduced when increasing the extract thread pool size.

By analyzing the impact on the GPU memory usage (Figure 5.5b), we observe that it increases as we allocate more threads to the extract thread pool, and it remains constant during the application execution. The GPU utilization for all thread pool sizes is between 35% and 60% most of the time, while the GPU power draw is between 50 Watts and 80 Watts. As the GPU memory usage, the system memory usage (Figure 5.5c) of the Docker container running the Pl@ntNet Engine also increases with the extract thread pool size.

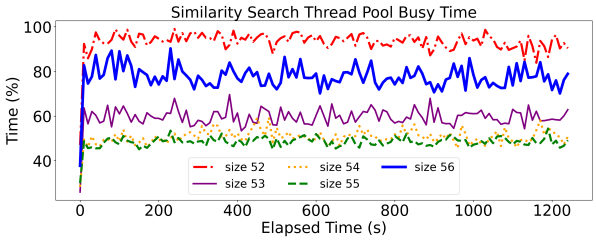
Lastly, the extract thread pool busy time (Figure 5.4c) is 100% during the whole application execution for thread pool sizes of 5, 6, and 7, and between 80% and 100% for sizes 8 and 9. This explains the higher and lower values of the *wait-extract* times observed in Figure 5.4b. For the similarity search (Figure 5.4d), the thread pool busy time is between 80% and 100% for a size of 8 and 9. For the 5, 6, and 7 thread pool sizes, it is 50%, 55%, and 60% busy on average, respectively. This also explains the higher values of *wait-simsearch* for sizes 8 and 9 compared to 5, 6, and 7 in Figure 5.4b.



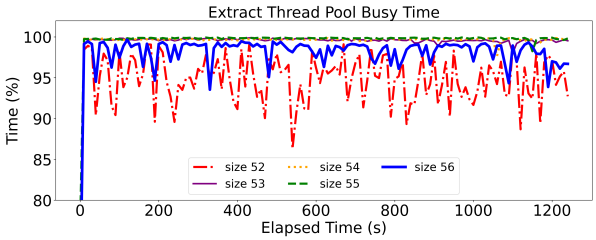
(a) user response time.



(b) processing time.



(c) simsearch pool busy time.



(d) extract pool busy time.

Figure 5.6 – Impact of similarity search thread variability.

Analyzing thread pool variations in the similarity search task

Following our analysis, Figure 5.6 shows the impact of the thread pool size for similarity search on: (*a*) user response time and (*b*) processing time. Besides, in Figure 5.6c and Figure 5.6d, we show the thread pool busy time for the similarity search and extract thread pools, respectively.

In Figure 5.6a, the preliminary optimum configuration with 53 threads may be increased to 55 threads to reduce by about 4% the user response time. Regarding the processing time (Figure 5.6b), the *simsearch* task time confirms what was observed with the user response time. That is, adding more than 55 threads is not worth to decrease its execution time.

Figure 5.6c shows the correlation of the similarity search pool busy time with the *simsearch* task time observed in Figure 5.6b and explains its variation. Using 52 threads, it is busy between

Table 5.3 – Comparison of the three Pl@ntNet configurations.

Thread pool	**baseline**	**preliminary optimum**	**refined optimum**
HTTP	40	54	54
Download	40	54	54
Extract	7	7	**6**
Simsearch	40	53	53
User response time	**2.657** ($\pm$0.0914)	**2.484** ($\pm$0.0912)	**2.476** ($\pm$0.0826)

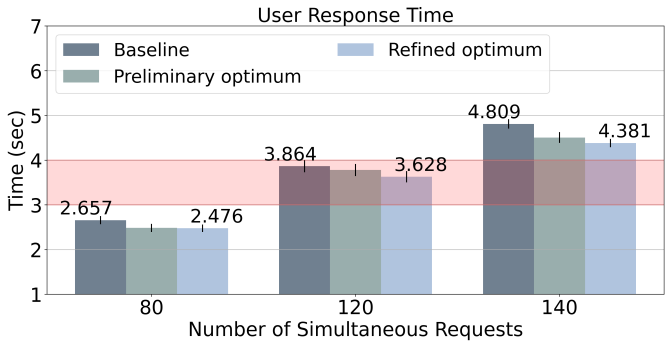


Figure 5.7 – User response time: baseline vs. optimums.

90% and 100%, while for 53 to 55, it is below 60% and increases to about 80% with 56 threads. The impact of the similarity search thread pool variation on the extract task (Figure 5.6b) can be explained by Figure 5.6d. Lower times in *wait-extract* for sizes 52 and 56 is due to a busy time between 90% and 100%. For sizes from 53 to 55, the busy time is 100%.

Exploring the refined optimum configuration

Since we observed a lower user response time after analyzing the impact of variations of the *extract* and *simsearch* thread pool configurations on the user response time, we exploit this configuration (named ***refined optimum***) with all the previously defined workloads. As presented in Table 5.3 and Figure 5.7, we observed even better results for all workloads.

The refined optimum presents the best results for all workloads, outperforming both the baseline and preliminary optimum. Compared with the baseline, the difference between configurations varied with the workloads as follows: from 6.9% to 7.2%; from 2.2% to 6.3%; and from 6.7% to 9.8% for 80, 120, and 140 simultaneous requests, respectively.

5.4 Reproducibility and Artifact Availability

Our methodologies are aligned with the Open Science [84] goal to make scientific research processes more transparent and results more accessible. As presented, our experiment methodology provides guidelines to systematically define the whole experimental environment, such

as the testbed resources, the network configurations, and the execution logic of the application workflow. Furthermore, our optimization methodology guides users to manage the optimization cycle of complex workflows, such as: (*Phase-I*) defines the optimization problem and the application-related parameters to optimize; (*Phase-II*) defines the optimization techniques and search algorithms; and (*Phase-III*) provides access to the optimization results.

The whole optimization cycle is set up through a *User-defined Optimization* class. This class is designed to be easy to use and understand, and it can be easily adapted to different optimization problems (find out more in the documentation Web page [225]). At the end of each optimization cycle, E2Clab provides an archive of the generated data. The archive consists of data from *Phases I and II*, which is needed to allow other researchers to reproduce the results. The access to the experimental artifacts, the definition of the experimental environment, and the experimental results are publicly available at [74].

Conclusion

This chapter presented the validation of our methodologies **at a large scale on 42 nodes of the Grid'5000 testbed**. It shows how E2Clab can be used to analyze and optimize the performance of the **Pl@ntNet botanical application**, used by more than 10 million users in 180 countries. We highlight that the analysis presented in this chapter, backed by our methodologies, **helps to understand** how variations in the thread pool configuration of the Pl@ntNet engine impact the processing times (user response time and identification processing steps) by correlating them with resource usage.
Furthermore, this analysis **helps to improve the performance of the application** by supporting 35% more simultaneous users and presenting a smaller *user response time* for different workloads (80, 120, and 140 simultaneous requests) and 30% less GPU memory utilization, when compared to the baseline.
The following chapter explores the efficient **provenance capture** of Edge-to-Cloud workflows for assisting users to **understand** the application performance trade-offs. It illustrates E2Clab with a **synthetic application** deployed on **multiple scientific testbeds** (*e.g.,* experiments combining Edge and Cloud resources).

Part III

Facilitating Reproducibility and Replicability of Edge-to-Cloud Workflows

Chapter 6

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

Contents

The third part of this thesis focuses on facilitating the reproducibility and replicability of performance trade-offs of Edge-to-Cloud workflows. In particular, we show how E2Clab can be applied to support other domains that are crucial for reproducibility, such as workflow provenance capture and collaborative environments for replicating Edge-to-Cloud experiments.

This Chapter introduces the ProvLight tool to enable efficient provenance capture in IoT/Edge devices. Then, it presents the integration of ProvLight into the E2Clab framework for capturing provenance data of Edge-to-Cloud workflows. Finally, it evaluates Provlight and shows that it outperforms existing solutions.

6.1 The Need for Provenance Capture of Edge-to-Cloud Workflows

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** [132]. **Capturing provenance data during workflow execution** helps users in tracking inputs, outputs, and processing history, allowing them to steer workflows precisely [207].

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?* [*201, 208*] or (*ii*) *Retrieve the hyperparameters which obtained the 3 best accuracy values for model* m*?* [*160, 208*]. Answering such queries helps to analyze hyperparameter values related to the training stages and to adjust them for better-quality results.

Overhead in provenance systems is a critical problem that must be assessed [109]. Many other contributions in provenance systems evaluate the overhead, such as [206, 202]. 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 [25], leading database researchers discussed the challenges of deploying services considering disaggregation and high heterogeneity of resources in hybrid cloud infrastructures. In [200], the authors describe challenges related to capturing provenance on the Edge-to-Cloud Continuum.

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.

We make the following contributions:

1. **A performance evaluation of the existing provenance systems** (*e.g.*, DfAnalyzer, ProvLake, PROV-IO, and Komadu) when capturing provenance **in IoT/Edge devices** (Section 6.2).
2. **A novel workflow provenance capture approach tailored for resource-limited IoT/Edge devices** that addresses the limitations found in state of the art (Section 6.3). **ProvLight is an open-source implementation** of this approach (available at [162]), following the W3C PROV-DM recommendations.
3. **An integration of ProvLight within the E2Clab framework**. This enables **provenance data capture across the Computing Continuum** for hybrid workflows deployed on 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 6.4). This integration with E2Clab is an open-source tool available at [75]. We highlight that **ProvLight may easily integrate into other deployment and performance optimization systems**.
4. **A large-scale experimental validation** of ProvLight with synthetic workloads **on 64 real-life IoT devices** (from FIT IoT LAB [4] testbed) and **Cloud resources** (from Grid'5000 [33] 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 6.5).

6.2 Limitations of Existing Provenance Systems

The main state-of-the-art provenance systems were designed to run on Cloud/HPC infrastructures. 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 [202], ProvLake [206], and PROV-IO [105]. We also include Komadu [216] in our analysis because the authors of the systems mentioned earlier compare with it.

Since our main goal is to enable the efficient provenance capture of workflows running on IoT/Edge computing resources, 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.

6.2.1 Experimental Setup

Selected provenance systems. Due to the limitations of the PROV-IO and Komadu systems, shown in Table 6.4, they were excluded from our performance analysis. We choose ProvLake

Table 6.1 – 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

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 [208], but this system version is not available as open-source.

Performance metrics. The main tracked metric is the *capture time overhead*, which refers to the relative difference in workflow execution time with and without data capture. To increase the accuracy of the results for each provenance system and each synthetic workload, we repeat the experiment 10 times. All results are presented as the mean of the ten evaluations followed by their respective 95% confidence interval.

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 [185]; $\leqslant$ 4% for I/O-centric workflows [105]; 4% for AI model training [201]; $\leqslant$ 12% for security applications [153]; 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 [141], real-time monitoring in smart energy grids [2], and virtual and augmented reality [134], 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.

Synthetic workloads. 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 [206, 202], we decided to

focus our analysis on synthetic workload configurations. Such configurations are based on real-life workloads [242, 133, 113], and we refined the configuration space of our workloads with preliminary experiments on real-life edge devices.

Table 6.1 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 FL 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 FL 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) [208]. Other transformations include data preparation and the evaluation of the trained model.

To generate our synthetic workload, we consider 100 tasks. In the FL 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).

Hardware. Each workload configuration runs on a single A8-M3 [112] 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 [4]. We instrument the synthetic workloads (code available at [161]) with the capture libraries provided by ProvLake and DfAnalyzer systems. The capture libraries transmit the captured data to the provenance system running on a remote Cloud/HPC server [91] (Intel Xeon Gold 5220, 2.20GHz, 18 cores; 96GB RAM; Ethernet) available at the Grid'5000 [33] testbed.

Table 6.2 – Capture overhead of ProvLake and DfAnalyzer.

overhead level	**low** $\leqslant 3\%$	**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 duration (s)	**0.5**	**1**	**3.5**	**5**

Table 6.3 – ProvLake: impact of bandwidth and grouping strategy on the capture overhead.

grouping data captured	**Bandwidth 1Gbit**		**Bandwidth 25Kbit**	
0	59.49% ±0.09	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**

6.2.2 Overhead Analysis

Table 6.2 presents the **capture time overhead** of ProvLake and DfAnalyzer **in IoT/Edge devices**, and Table 6.3 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 6.2 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 6.3, we observe low overhead (<3%) when grouping 50 messages for a

Table 6.4 – 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.

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.

6.2.3 Design-level Limitations of Existing Systems

Table 6.4 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 [145], CoAP [50], AMQP [8], UDP [218], RPL [183], 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 [144, 230, 93].

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.

6.3 ProvLight Design

This section introduces ProvLight, a tool [162] 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. Section 6.3.1 presents the ProvLight provenance model. Next, the architectural details are given in Section 6.3.2, while Section 6.3.3 describes its implementation.

6.3.1 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 ensuring these captured data are compatible with W3C PROV-based workflow provenance systems, such as ProvLake, DfAnalyzer, PROV-IO, among many others. Table 6.5 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).

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.

6.3.2 Architecture

Figure 6.1 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 Df-

Table 6.5 – 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.	

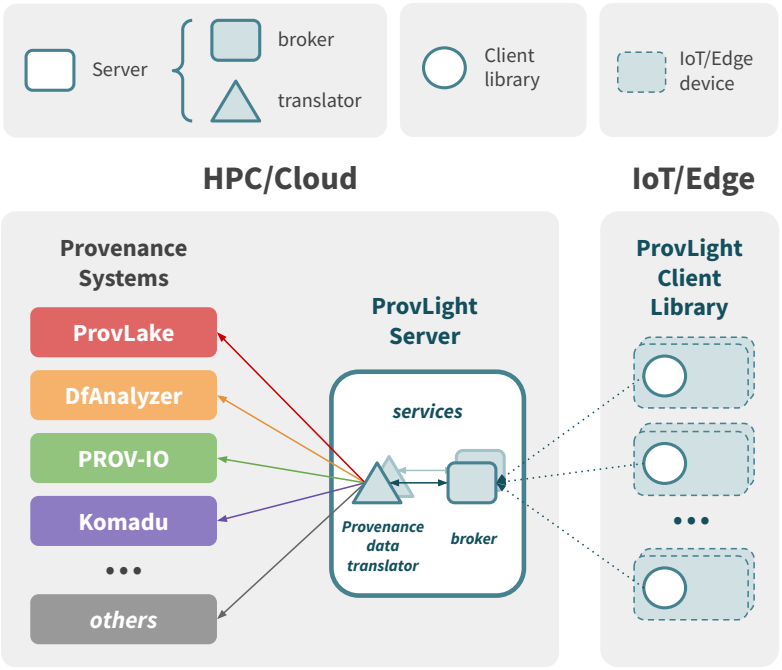


Figure 6.1 – ProvLight Architecture.

Analyzer, 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 6.1. Table 6.6 summarizes how the ProvLight architecture design differs from the systems analyzed in Section 6.2.

This integration may be achieved by using:

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 [210]). 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.

Table 6.6 – 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

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 6.5). This library 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.

6.3.3 Implementation

Server

The *Broker* is implemented based on the Eclipse RSMB server [184] (Really Small Message Broker). RSMB builds on top of Mosquitto [78] codebase 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 [162], 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 [163] 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 [172]).

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.

Capturing Provenance using ProvLight

Listing 6.1 illustrates an example of application code instrumentation with the ProvLight library highlighted in blue color. Lines 7, 8, and 24 instantiate the workflow, start, and finalize it, respectively. Line 17 instantiates a task, linking it to the *workflow*, input data *derivation*, and dependent task. Lines 19 and 22 capture data from the initialization and finalization of the task.

Before starting a task, line 18 instantiates *Data* and adds it as input data (line 19) to the task. Following the same logic, line 21 instantiates and adds the output data from the task. The *begin*() and *end*() methods of *Workflow* and *Task* transmit the captured data over the network to the *broker*. Finally, line 20 is where the workflow task runs.

6.4 Provenance Capture of Edge-to-Cloud Workflows

This section presents the integration of ProvLight as a key system in the E2Clab [176] 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 6.2 shows the extended E2Clab architecture with the new components highlighted in red color.

```
from provlight.workflow import Workflow
from provlight.task import Task
from provlight.data import Data

attributes = 100, chained_transformations = 5, number_of_tasks = 100
# Application Workflow
workflow = Workflow(1)
workflow.begin()
# Tasks and data derivations
data_id = 0, previous_task = []
in_data = {'in': [1 for _ in range(attributes)]}
out_data = {'out': [2 for _ in range(attributes)]}

for transf_id in range(chained_transformations):
    for task_id in range(int(number_of_tasks/chained_transformations)):
        data_id += 1
        task = Task(transf_id-task_id, workflow, transf_id, dependencies=
    previous_task)
        data_in= Data(in_{data_id}, workflow.id, in_data)
        task.begin([data_in])
        #### ADD YOUR TASK HERE ####
        data_out= Data(out_{data_id}, workflow.id, out_data)
        task.end([data_out])
        previous_task = [task.id]
workflow.end()
```

Listing 6.1 – ProvLight: user-defined provenance capture.

6.4.1 Provenance Manager

We design a new manager named *Provenance Manager*. Figure 6.2 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/Edge 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).

We highlight that in addition to the characteristics of the provenance systems analyzed in Table 6.4, and due to ProvLake being proprietary within IBM, while DfAnalyzer is open source [67], in this work, we decide to use DfAnalyzer. Note that, 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 respective provenance system and use their APIs.

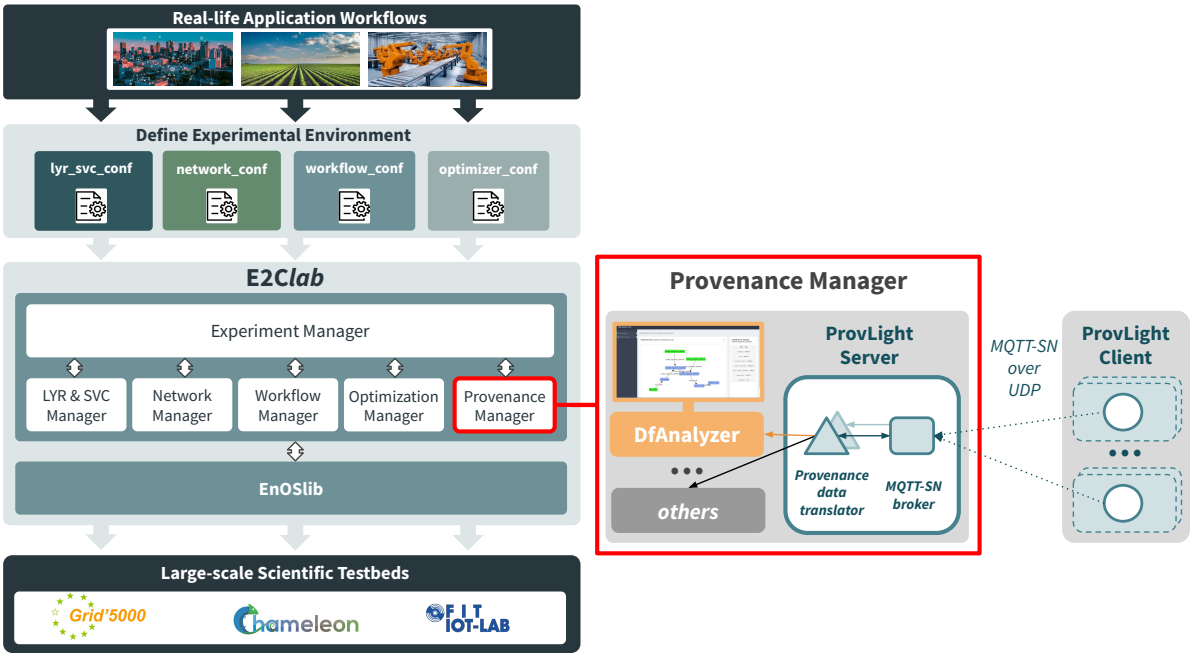


Figure 6.2 – Extended E2Clab: Provenance Data Manager.

6.4.2 Provenance Capture

Through the E2Clab framework, users may easily enable provenance data capture across the Edge-to-Cloud continuum through simple configuration files, as illustrated in Listing 6.2. Listing 6.2 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 7 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 6 setups the provenance data capture (the *ProvenanceManager* service). After that, users may instrument their application code to capture data, as presented in Listing 6.1.

The *ProvenanceManager* service is presented in Listing 6.3. Line 10 starts a Docker [70] container with the DfAnalyzer provenance system. While line 16 refers to the ProvLight container exposed at port 1883 to allow clients to send their provenance data. Then, it translates and sends data to DfAnalyzer at port 22000. Finally, DfAnalyze publishes the captured data on its Web interface at workflow runtime. We highlight that the *ProvenanceManager* service may be easily plugged into other provenance systems by just using their respective Docker containers in line 10 and extending the provenance data translator.

```yaml
environment:
  g5k:
    cluster: gros
  iotlab:
    cluster: grenoble
provenance:
  name: ProvenanceManager
layers:
- name: cloud
  services:
  - name: Server
    environment: "g5k"
    quantity: 1
- name: edge
  services:
  - name: Client
    environment: "iotlab"
    archi: a8
    quantity: 10
```

Listing 6.2 – E2Clab: provenance of Edge-to-Cloud workflows.

```python
from e2clab.services import Service
import enoslib as en

class ProvenanceManager(Service):
   def deploy(self):
      self.deploy_docker()
      with en.actions(roles=self.roles) as a:
         # DfAnalyzer
         a.docker_container(
            name="dfanalyzer",
            image="vitorss/dataflow_analyzer",
            published_ports="22000:22000"
         )
         # provlight
         a.docker_container(
            name="provlight",
            image="username/provlight",
            published_ports="1883:1883"
         )
      return self.register_service(port=[1883])
```

Listing 6.3 – E2Clab: Provenance Manager implementation.

6.5 Evaluation

This section aims 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 in the Cloud?*** We answer these questions in subsections 6.1—6.4 and 6.5, respectively, by comparing ProvLight against ProvLake and DfAnalyzer.

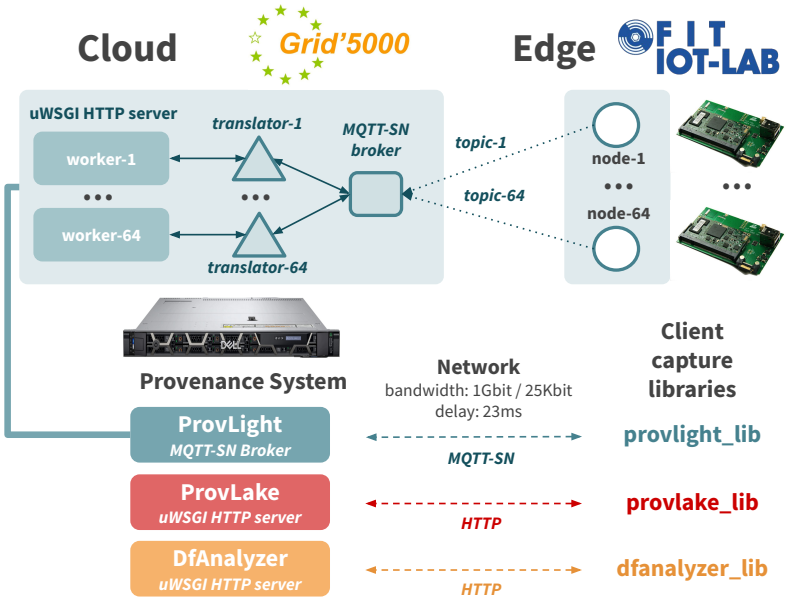


Figure 6.3 – Experimental setup: more details in Section 6.2.1.

The *main performance metric* is the capture overhead (relative difference of the workflow execution with and without data capture) 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 6.2.1**, with synthetic workloads generated based on the FL use case. The deployment is illustrated in Figure 6.3. Results in Figure 6.4 are the mean of 10 runs with their 95% confidence interval.

6.5.1 Capture Time Overhead

Table 6.7 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.**

Similarly to Table 6.3, Table 6.8 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 6.3.

Table 6.7 – Capture overhead in IoT/Edge devices.

Attributes per task	System	Capture Overhead (%)			
10	ProvLake	56.9% ±0.08	29.9% ±0.29	8.56% ±0.01	6.02% ±0.01
	DfAnalyzer	39.8% ±0.06	21.2% ±0.34	6.12% ±0.07	4.26% ±0.01
	ProvLight	1.45% ±0.01	1.02% ±0.01	0.31% ±0.01	0.23% ±0.01
100	ProvLake	57.3% ±0.10	30.1% ±0.40	8.57% ±0.01	6.04% ±0.04
	DfAnalyzer	40.5% ±0.20	21.3% ±0.06	6.12% ±0.01	4.31% ±0.01
	ProvLight	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 6.9 – 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**

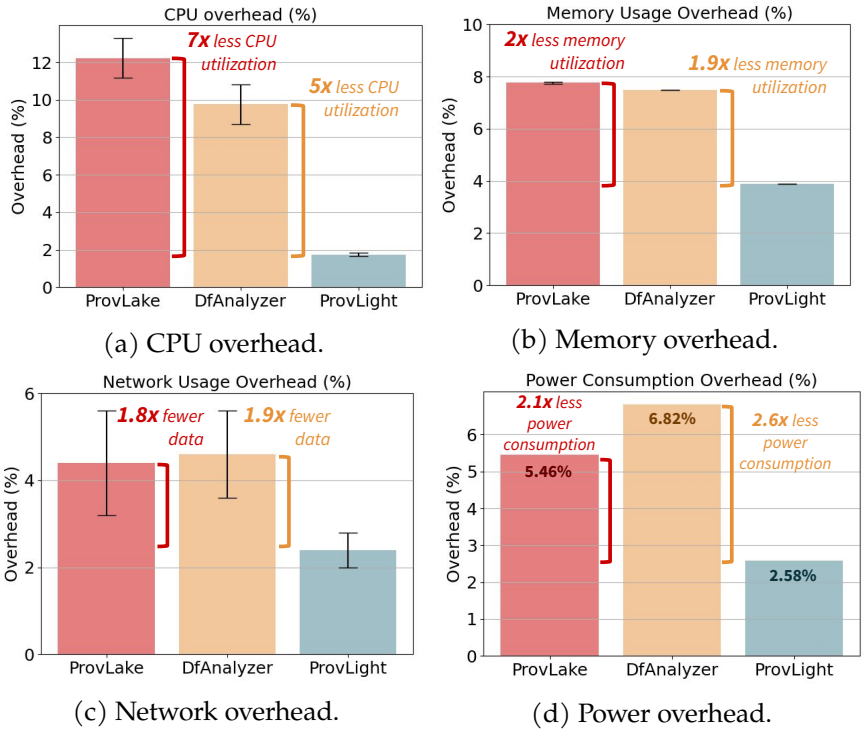
Scalability analysis. Table 6.9 presents the capture time overhead of ProvLight when scaling the number of IoT/Edge devices and considering 100 tasks of 0.5s each and 100 attributes per task. We scale the scenario with 8, 16, 32, and 64 devices capturing provenance data in parallel and sending the data to the cloud server.

As illustrated in Figure 6.3, 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 [34] 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.

Table 6.8 – How do bandwidth variations and the grouping strategy impact the capture overhead?

grouping		Bandwidth 1Gbit				Bandwidth 25Kbit			
data captured		ProvLake	ProvLight	ProvLake	ProvLight	ProvLake	ProvLight	ProvLake	ProvLight
0	0.5	59.49% ±0.09	2.10% ±0.01	30.1% ±0.27	1.10% ±0.01	321% ±1.05	2.00% ±0.01	161% ±1.14	1.04% ±0.01
	1	6.84% ±0.02	1.37% ±0.01	3.57% ±0.20	0.75% ±0.01	102.5% ±3.89	1.37% ±0.01	49.8% ±2.92	0.74% ±0.01
10	0.5	3.87% ±0.01	1.32% ±0.01	1.98% ±0.01	0.72% ±0.01	100.8% ±3.78	1.34% ±0.01	51.1% ±1.03	0.73% ±0.01
	1	2.37% ±0.01	1.31% ±0.01	1.27% ±0.01	0.72% ±0.01	95.0% ±0.10	1.31% ±0.01	43.2% ±0.28	0.72% ±0.01
20	0.5								
	1								
50	0.5								
	1								
task duration (s)									



(a) CPU overhead.

(b) Memory overhead.

(c) Network overhead.

(d) Power overhead.

Figure 6.4 – Provenance data capture overhead with respect to: CPU, memory, network usage, and power consumption.

6.5.2 CPU and Memory Overhead

Figures 6.4a and 6.4b present the CPU and memory overhead for capturing provenance data with ProvLake, DfAnalyzer, and ProvLight (from left to the right). Regarding the CPU overhead, **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.

6.5.3 Network Usage Overhead

As presented in Figure 6.4c, **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.

Table 6.10 – 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**

6.5.4 Power Consumption Overhead

Finally, results in Figure 6.4d (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.

6.5.5 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 [91] available in Grid'5000). Experiment results in Table 6.10 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.

6.6 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.

6.6.1 ProvLight Design Choices Impact on Performance

As presented in Table 6.6, the combination of ProvLight design choices on the server and client sides contributed to the low capture overhead. The ProvLight client library keeps the con-

nection 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 6.5), 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.

6.6.2 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 [160] and [201]. 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 6.1.

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 6.2.

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.

6.6.3 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) of workflows executed in IoT/Edge devices.

As presented in Subsection 6.3.2, 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 6.4: (*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.

6.6.4 Reproducibility and Artifact Availability

The experimental evaluations presented in this work follow a rigorous methodology [176] 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 [161].

Conclusion

The provenance data capture approach proposed in this chapter, and implemented in ProvLight, has proven **efficient in workflows executed on resource-limited IoT/Edge devices**. 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.

We have validated our capture approach with synthetic workloads on real-life IoT/Edge devices in the FIT IoT LAB testbed. Experiments comparing ProvLight with ProvLake and DfAnalyzer show that it **outperforms** these systems. ProvLight is 26-37x faster in capturing provenance data; uses 5—7x less CPU, 2x less memory, transmits 2x less data, and consumes 2—2.5x less energy.

The following chapter explores the **cost-effective reproducibility** of experiments. It proposes a collaborative environment and illustrates with a **real-life application** deployed on the **Grid'5000 and FIT IoT LAB testbeds**. It shows how **independent researchers** can replicate the same experiments on the **Chameleon Cloud and CHI@Edge testbeds**.

Chapter 7

Cost-effective Reproducibility and Replicability of Edge-to-Cloud Experiments

Contents

One way of allowing users to benefit from frameworks like E2Clab and to further contribute to them is through the use of collaborative environments. In this chapter, we introduce KheOps, an approach that aims to allow the cost-effective repeatability, reproducibility, and replicability of Edge-to-Cloud experiments on large-scale scientific testbeds. It illustrates KheOps with a real-life application and shows how it helps to replicate results accurately.

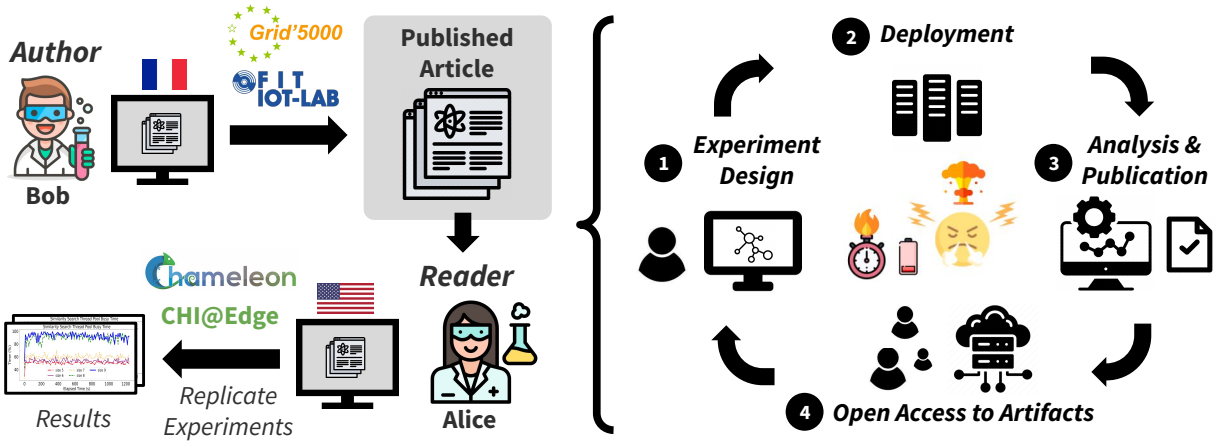


Figure 7.1 – Processes for reproducing and replicating experiments regarding the authors and readers point of view.

7.1 Requirements for Reproducible and Replicable Experiments

Systematically performing experiments on the Computing Continuum to enable their reproducibility and replicating their performance trade-offs are inherently difficult [103]. Figure 7.1 illustrates such processes.

Let us consider the case of a group of researchers who execute their experiments on French scientific testbeds such as Grid'5000 [33] (providing Cloud/HPC servers) and FIT IoT LAB [4] (providing IoT/Edge devices), and want to publish their results in an article. Next, the readers want to replicate the experiments on American testbeds such as the Chameleon Cloud [121] and CHI@Edge [119].

These processes compel a lot of effort, are time-consuming, and bring many technical challenges for both sides. For instance, also depicted in Figure 7.1, they require:

1. following methodologies to design the experiments systematically and to reconcile many application requirements or constraints in terms of energy consumption, network efficiency, and hardware resource usage;
2. configuring systems and networks and deploying applications on testbeds for large-scale evaluations;
3. analyzing, repeating experiments, and publishing results;
4. providing open access to the experiment artifacts in a public and safe repository.

Given such complexities, researchers end up not following rigorous methodologies for supporting the reproducibility of the experiments, as observed in our previous survey [175] and summarized in Figure 7.2. Consequently, it makes it hard for other researchers to replicate the published studies.

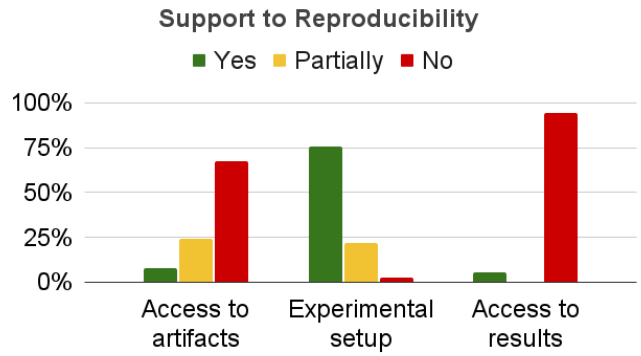


Figure 7.2 – Support to the reproducibility of Edge-to-Cloud experiments provided by the selected studies in our survey [175].

Let us sum up the associated requirements in this context. To enable **reproducible** experiments on the Edge-to-Cloud continuum, the requirements (a-REQ) of the authors of the experiments can be described as follows:

a-REQ 1. Execute experiments on heterogeneous computing resources (*e.g.*, IoT/Edge and Cloud/HPC infrastructures).

a-REQ 2. Systematically describe and explain the experimental processes and their reasoning.

a-REQ 3. Efficiently configure the experimental infrastructure and express topologies in repeatable ways.

a-REQ 4. Easily share the experiment artifacts in a public and safe repository.

At the same time, to enable the **replicability** of the experiments, the readers of an article describing those experiments have the following requirements (r-REQ):

r-REQ 1. Find and access the experiment as simple as finding and reading its paper.

r-REQ 2. Perform the experiment, not just read about it.

r-REQ 3. Answer not just to the *"What"* question (What the experiment does?), but also the *"Why"* (Why did authors set it up that way?) and *"How"* (How did authors connect machines/devices?)

r-REQ 4. Efficiently configure the experimental infrastructure to reduce the time spent satisfying all the experiment requirements.

In this Chapter, we study the challenges of cost-effectively reproducing and replicating Edge-to-Cloud experiments. Cost-effective means to allow authors and readers to efficiently fulfill their experimental requirements as previously described. This calls for practical solutions beyond the state-of-the-art.

Our main objective is to provide a collaborative environment and methodology that supports reproducible Edge-to-Coud experimentation between different open testbeds such as Grid'5000, FIT IoT LAB, Chameleon, *etc.*, equipped to deal with IoT/Edge and Cloud/HPC resources which are fundamental to reproducibility [118]. We propose the following main contributions:

1. **A study of the characteristics of the main state-of-the-art collaborative environments** (*e.g.*, Google Colab, Kaggle, and Code Ocean) for enabling reproducible experiments. Their **main limitations in the context of Computing Continuum research** are discussed in Section 7.2.
2. **A novel collaborative environment to enable reproducible Edge-to-Cloud experiments** (Section 7.3). This approach, named KheOps, allows researchers to reproduce and replicate Edge-to-Cloud workflows cost-effectively. KheOps core elements are: (1) a portal for sharing experiment artifacts; (2) a notebook environment for packaging code, data, environment, and results; and (3) a **multi-platform** experimental methodology for deploying experiments on heterogeneous resources from the IoT/Edge (FIT IoT LAB and CHI@Edge) to the Cloud/HPC Continuum (Grid5000 and Chameleon). We highlight that KheOps may be integrated with other large-scale scientific testbeds.
3. An **experimental validation** of the proposed approach with a **real-world use case deployed on real-life IoT/Edge devices and Cloud/HPC systems**. The evaluations show that KheOps helps: (1) authors to perform reproducible experiments on the **Grid5000 + FIT IoT LAB** testbeds, and (2) readers to cost-effectively replicate authors experiments on the **Chameleon Cloud + CHI@Edge** testbeds, and **obtain the same conclusions with high accuracies**, >88% for all performance metrics (Section 7.4).

7.2 Limitations of Existing Collaborative Environments

We briefly discuss the limitations of state-of-the-art collaborative environments, with a focus on the specific challenges of the Computing Continuum.

Google Colab [96] Mainly used by the AI community (more than 50K users), it is a ready-to-use Jupyter notebook service. Colab notebooks are stored in the *.ipynb* open-source Jupyter notebook format [97], and come with the most popular AI libraries and frameworks installed (*e.g.*, Scikit-Learn [156], TensorFlow [1], PyTorch [154], *etc.*) and allow users to run python code through the browser. It is typically used for machine learning, data analysis and education.

Colab is popular because it allows users to share Jupyter notebooks without having to download, install, or run anything. Besdides, it provides free access to very expensive computing resources such as GPUs and TPUs. Colab permits multiple users to collaborate on the same notebook. Sharing datasets, ML models, pipelines, and notebooks on AI Hub [5] is also possible

(more than 167 notebooks). Its GitHub integration allows users to quickly open GitHub-hosted Jupyter notebooks in Google Colab.

Kaggle [114] This is a data science and AI platform that offers a customizable Jupyter notebook environment. Kaggle is a subsidiary of Google and, like Colab, it provides free access to GPUs as well as a repository of community-published (more than 10.3 million users) datasets (more than 50K public datasets) and code (*e.g.*, machine learning code) with more than 400K public notebooks.

Kaggle is integrated with AI Hub and is popular in the data science and machine learning communities. Kaggle is also well-known for promoting Community Competitions in machine learning at no cost.

The main differences [98] between Colab and Kaggle are:

1. Kaggle allows collaboration with other users on its Web site, while Colab allows collaboration with anyone using the notebook link;

2. Kaggle has many data sets that users can use directly (*e.g.*, notebooks already set up with Kaggle databases [115]), while in Colab setting up notebooks with Google Drive [53] or managing files [52] (*e.g.*, to load data sets, files, and images) requires extra work;

3. Kaggle creates a history of notebook commits that we can be reviewed.

Code Ocean [49] Designed according to FAIR [229] (*i.e.*, Findable, Accessible, Interoperable, and Reusable), Code Ocean aims to make scientific work reproducible. It introduces the concept of Compute Capsule, which refers to Docker [69] containers composed of code, data, environments, and results. Capsules provide ready-to-use tools such as Git, Jupyter, RStudio, among others. Its integration with Git allows users to save changes on capsules and then commit them with just one click.

Furthermore, users can easily share the link of a capsule and grant permission. Code Ocean provides scalable compute and storage resources hosted on Amazon Web Services. Resources used by capsules are scaled out when the demand exceeds the machine capacity. Finally, Code Ocean provides a public Capsule Repository [51] with more than 1K research capsules. It allows authors of an article to incorporate capsules into the submission process via a Hub publishing API.

Table 7.1 – Limitations of Existing Collaborative Environments.

Limitation	Google Colab	Code Ocean	Kaggle
Resource heterogeneity	CPU, disk, and memory limits; GPU types available; no access to IoT/Edge devices;	experiments run on AWS virtual machines; no access to IoT/Edge devices;	limits CPU, GPU, and TPU access; does not support IoT/Edge devices;
Large-scale experiments	limits sessions to 12 hours; paid access to multiple computing resources.	limits access to 10 compute hours; paid access to multiple computing resources.	limits execution time to 12 hours; paid access to Google Cloud Services.
Repeatability, Reproducibility, Replicability	hard to repeat and reproduce experiments on the same hardware: resource availability varies over time and usage limits fluctuate. Replicability in different infrastructures (*e.g.*, beyond Google machines) is not straightforward.	lacks support for the reproducibility of distributed experiments. Computing and storage resources are available in AWS virtual machines in the clients virtual private cloud. Hard to replicate experiments in different infrastructures.	lacks support for the repeatability and reproducibility of distributed experiments. Computing resources vary over time and hence between accesses. Replicability in different infrastructures is not easy to set up.

Despite these systems being widely used by the AI and data science communities, they present some limitations that hinder their adoption for Computing Continuum research. Table 7.1 summarizes these limitations in terms of:

1. access to heterogeneous IoT/Edge and Cloud/HPC computing resources;
2. support for large-scale experimental evaluations;
3. repeatability and reproducibility of experiments on the same hardware setup, and replicability on different infrastructures.

In summary, collaborative environments lack support for providing access to heterogeneous resources (*e.g.*, Edge-to-Cloud); performing experiments at large-scale; and achieving the reproducibility of experiments on the same hardware setup. Hence, the need for novel approaches for reproducible evaluations of workflows targeting the characteristics of the Computing Continuum.

7.3 Kheops Design

This section introduces KheOps, a collaborative environment for the cost-effective reproducibility and replicability of Edge-to-Cloud experiments. KheOps is designed to meet the experimental requirements of both authors and readers as presented earlier.

7.3.1 Architecture and Implementation

Figure 7.3 presents the architecture of KheOps, which consists of three main components: (*i*) Trovi sharing portal; (*ii*) Jupyter environment (JupyterHub service and JupyterLab server); and (*iii*) E2Clab framework (multi-platform experiment methodology). Next, we present the integration details of these components, and briefly describe their main roles.

Experiment Repository

KheOps uses Trovi to share research artifacts such as packaged experiments. These artifacts may be publicly available to allow others to recreate and rerun experiments. Trovi provides a REST API to manage experiment artifacts and integrate them with other systems. The JupyterHub in KheOps uses the Trovi REST API to download artifacts and launch them in the JupyterLab server.

Artifacts hosted in Trovi can also provide references to repositories like container registries (*e.g.*, DockerHub [227]), multipurpose repositories (*e.g.*, Zenodo [240]), code repositories (*e.g.*, Github [95]), and among others.

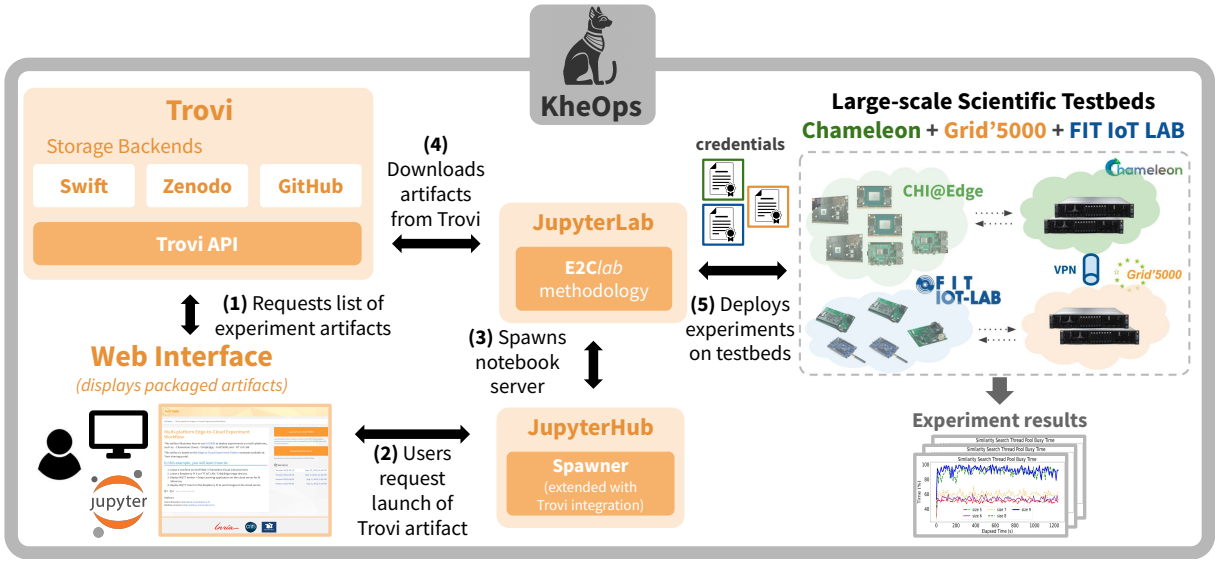


Figure 7.3 – KheOps architecture and experimental workflow.

Notebook Environment

Following our previous work [9] on integrating experiment workflows with Jupyter notebooks, we extend JupyterHub to authenticate users and to download (using the Trovi REST API) the experiment artifacts available at Trovi. We also extend JupyterLab to allow users to easily share their experiments in Trovi. Furthermore, JupyterLab is set up with the E2Clab framework as an experimental methodology.

The JupyterLab is packaged with code, data, environment configurations, and experiment results. Its notebooks (file extension *.ipynb*) allow users to run experiments step-by-step by combining text (*e.g.*, explaining the reasoning of the experiments: *What* parameters? *Why* these parameters? and *How* it was set up?) with executable code. Such notebooks are ready to use (*e.g.*, installed with required library/software), executed through a browser, and shared as a Trovi artifact.

Multi-testbed Experiment Methodology

KheOps uses the E2Clab methodology to deploy experiments on large-scale scientific testbeds such as Grid'5000, Chameleon Cloud, CHI@Edge, and FIT IoT LAB. Notebooks come with three main template files (*e.g.*, executable code cells in the notebook, presented in Listings 7.1 to 7.4) that users can benefit from to easily configure and adapt the deployment logic (*e.g.*, computing resources, network, and application execution) according to their experimental needs.

The first file, named *layers_services.yaml* and presented in Listing 7.1, allows users to lease IoT/Edge and Cloud/HPC resources. Through this file, users may also set up their applications

```yaml
environment:
  g5k: cluster: dahu
  iotlab: cluster: grenoble
layers:
- name: cloud
  services:
  - name: Server
    environment: g5k, quantity: 1
- name: edge
  services:
  - name: Client
    environment: iotlab, archi: rpi3, quantity: 5
```

Listing 7.1 – E2Clab: layers and services configuration.

```python
from e2clab.services import Service
import enoslib as en

class Server(Service):
   def deploy(self):
      with en.actions(roles=self.roles) as a:
        a.shell("pip3 install torchvision torch")
        a.shell("pip3 install pillow paho-mqtt")
      return self.register_service(port=[1883])
```

Listing 7.2 – E2Clab: user-defined service for the Cloud server.

and services as presented in Listing 7.2. Next, the *network.yaml* file (Listing 7.3) allows users to define delay, loss, and bandwidth between computing resources. Finally, the *workflow.yaml* file (Listing 7.4) guides users to define the experiment workflow through three main steps: *prepare* (*e.g.,* copy artifacts to remote nodes, install libraries, *etc.*), *launch* (*e.g.,* execute the application parts), and *finalize* (*e.g.,* backup results from remote nodes to the JupyterLab server).

E2Clab abstracts all the complexities of deploying and executing experiments across various testbeds. To do so, users need to add the credential files of the respective testbeds to their notebooks. Setting up a VPN is also supported as this may be required to enable the communication between different geographically distributed testbeds (*e.g.,* Chameleon in the USA and Grid'5000, FIT IoT LAB from France).

7.3.2 Experimental Workflow

In summary, the workflow for launching an experiment artifact on large-scale testbeds consists of 5 main steps. First, through a web interface, users can browse the list of experimental artifacts publicly available in Trovi (step 1). Selecting an artifact displays details such as the experiment description, the authors and contact information, and the artifact versions.

A *launch* button allows users to execute the artifact (step 2). This button redirects users to the JupyterHub service. After authentication, the request to launch the artifact is sent to the JupyterHub Spawner. Next, the Spawner spawns the JupyterLab server (step 3), and then it

```
networks:
- src: cloud, dst: edge
  delay: 150ms, rate: 25kbit, loss: 0.02
```

Listing 7.3 – E2Clab: network configuration.

```
- hosts: edge.*
  depends_on:
    conf_selector: cloud.server.*, grouping: round_robin, prefix: "server"
  prepare:
  - copy:
      src:{{working_dir}}/artifacts, dest: /
  launch:
  - shell: bash /edge_worker.sh edge_data 100 "{{server.ip}}" False
  finalize:
  - fetch:
      src:/tmp/predict.log, dest:{{working_dir}}/
```

Listing 7.4 – E2Clab: workflow configuration.

downloads experimental artifacts such as notebooks, code, and datasets, among others (step 4). The JupyterLab service is set up with the E2Clab framework as the experimental methodology. Finally, users can execute the code cells from the notebook to lease IoT/Edge and Cloud/HPC computing resources available on the testbeds, deploy and execute the application, and gather the experiment results (step 5).

Steps 2 to 4 are automatically executed. This is a one-click feature that allows users to have a ready-to-use environment for reproducing and replicating complex Edge-to-Cloud experiments cost-effectively. Note that the whole workflow requires only three clicks: selecting the experiment artifact (step 1), then launching it (steps 2 to 4), and executing it on the testbeds (step 5).

7.4 Evaluation

In this section, we show how KheOps can be used to analyze the performance of a real-life Edge-to-Cloud application deployed in the African savanna (illustrated in Figure 7.4). This application is composed of distributed Edge devices monitoring animal migration in the Serengeti region. Devices at the Edge collect and compress wildlife images, then the image is sent to the Cloud where the animal classification happens using a pre-trained Neural Network model. Finally, classified data helps conservationists to learn what management strategies work best to protect species.

The goals of these experiments are:

— to understand the impact on performance of **Cloud-centric** and **Hybrid** (**Edge+Cloud**) processing;

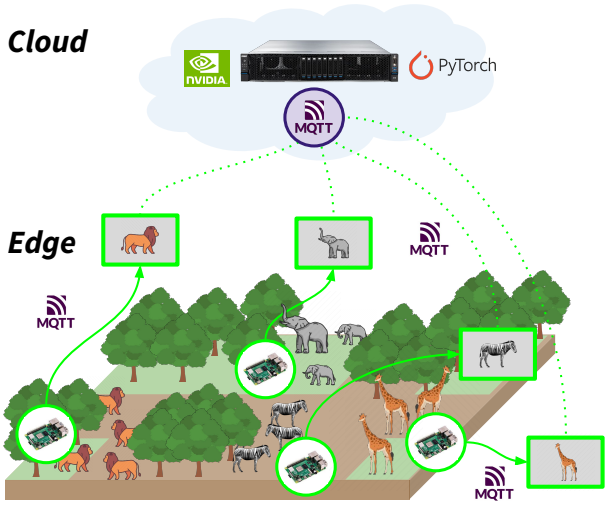


Figure 7.4 – Edge-to-Cloud application: monitoring animals migration in the African savanna.

— to show how authors of an article can benefit from KheOps to make their **experiments reproducible**;

— to show how readers of an article can leverage KheOps to **replicate the experiments** in an article (published using KheOps).

To reproduce the evaluations in this section, refer to [82].

7.4.1 Experimental Setup

Application performance metrics The main tracked metric is the *processing time*, which refers to the time required to: pre-process the image captured (*e.g.*, image compression on the Edge device); transmit the image to the Cloud server; and finally decompress the image and predict the animal through an AI model. In addition, we analyze the *amount of data transmitted* to the Cloud and the *resource consumption* (*e.g.*, CPU and memory) on the Edge device.

To increase the accuracy of the results, we measure the processing duration 100 times for each experiment, each time with a different image and an interval of 30 seconds (*i.e.*, Edge devices transmit images to the Cloud server every 30 seconds). The remaining metrics are captured using Dool (Dstat) [72] at application runtime. All results are presented as the mean followed by their respective 95% confidence interval.

KheOps replicability metric To measure how close/precise readers experiments are from authors experiments, we define the Replicability Accuracy ($Rep_{accuracy}$) metric. For assessing variability and error in results [193], a recommendation is to repeat the experiments multiple times to achieve narrower inferential error bars (i.e., confidence interval, standard deviation, *etc.*) [57]. The Replicability Accuracy metric is calculated as Equation 7.1:

$$Rep_{accuracy} = 1 - \left| \frac{min(x_{1A}, x_{2A})}{max(x_{1A}, x_{2A})} - \frac{min(x_{1R}, x_{2R})}{max(x_{1R}, x_{2R})} \right| \quad (7.1)$$

Ideally $Rep_{accuracy}$ would be close to 1. x_{iA} and x_{iR} refer to the application performance metric value obtained from authors and readers experiments, respectively. For instance, in Figure 7.5a, x_{1A} refers to the Cloud-centric bar and x_{2A} to the Edge + Cloud bar.

Workload Devices at the Edge transmit images (from the Snapshot Serengeti dataset [246] composed of millions of wildlife images collected annually) to the Cloud server that predicts animals using a trained MobileNetV3 Convolutional Neural Network model. We evaluate this workload considering two network configurations, 25Kbit and 15Kbit bandwidth, with a round-trip delay of 150ms.

Software On the Edge devices, we use zlib [164] Python library to compress images. MQTT [145] protocol transmits images to the Cloud. On the Cloud server, we use an MQTT broker to receive images, then zlib to decompress images, and finally PyTorch to predict animals.

Hardware The **authors** perform experiments on the following testbeds in France: Grid'5000 and FIT IoT LAB. On Grid'5000 (Cloud server), they use the dahu [58] machine equipped with an Intel Xeon Gold 6130 CPU 2.10GHz, 16 cores/CPU, 192GB of RAM, and Ethernet network. On FIT IoT LAB (Edge device), they use a Raspberry Pi 3 Model B [168] with four ARM Cortex-A53 1.2GHz processing cores, 1GB LPDDR2 memory, and 2.4GHz 802.11ac wireless LAN.

The **readers** replicate authors experiments on the following testbeds in the USA: Chameleon Cloud and CHI@Edge. On Chameleon CHI@TACC (Cloud server), they use the Skylake [54] machine equipped with an Intel Xeon Gold 6126 CPU 2.60GHz, 12 cores/CPU, 192GB of RAM, and Ethernet network. On CHI@Edge (Edge device), they use a Raspberry Pi 4 [169], with four BCM2711 Cortex-A72 processing cores running at 1.5GHz, 8GB LPDDR4 memory, and 2.4GHz and 5GHz 802.11ac wireless LAN.

7.4.2 How KheOps Helps Experiment Authors

Let us consider the requirements of the experiment authors (a-REQ) as introduced earlier.

a-REQ 1. Execute experiments on heterogeneous computing resources KheOps provides access to IoT/Edge devices and Cloud/HPC resources at large-scale, using the E2Clab methodology. Supported testbeds include (but are not limited to, as explained in Section 7.5.4): Grid'5000, FIT IoT LAB.

a-REQ 2. Systematically describe and explain the experimental processes and their reasoning Through Jupyter notebooks and the E2Clab configuration files, the authors describe and explain the experiment design choices such as the layers (*e.g.*, Edge and Cloud), the services (*e.g.*, the Edge client and the Cloud server), the network constraints, and the application workflow execution. This is done in Jupyter notebooks by combining text (explaining the configurations) followed by executable code (E2Clab configuration files).

a-REQ 3. Efficiently configure the experimental infrastructure and repeat the experiments All the complexities of configuring the Edge-to-Cloud infrastructure, such as leasing computing resources, mapping the application parts (*e.g.*, Edge and Cloud services), enforcing the network constraints, and executing the workflow are transparently handled by KheOps. The authors just need to define their experimental needs in the E2Clab configuration files. Repeating and adapting the experiments (*e.g.*, changing the network constraints) is easily done through E2Clab instrumentation.

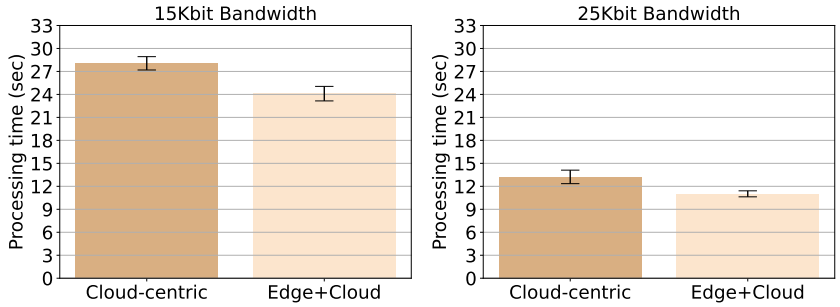
a-REQ 4. Easily share the experiment artifacts in a public and safe repository Through the Trovi and JupyterLab integration, authors can upload their artifacts to the Trovi sharing portal with a few clicks.

We discuss the experimental results from the authors perspective, using the three application performance metrics mentioned earlier.

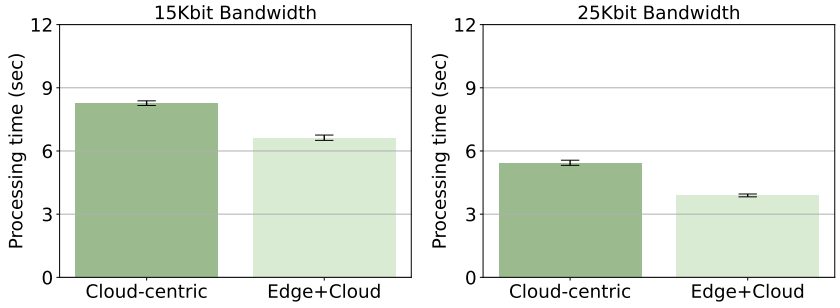
Impact of the network on the processing time

The authors define two sets of experiments. In the first one (Figure 7.5a), they fix the network bandwidth at 15Kbit and vary the processing approach between Cloud-centric and Hybrid (Edge + Cloud). In the second one (Figure 7.5b), they fix the bandwidth at 25Kbit for both processing approaches.

From the results, the authors observe that the Hybrid (Edge + Cloud) approach outperforms the Cloud-centric one for both network configurations. In the 15Kbit bandwidth setup, the processing time for the Cloud-centric is about 27 seconds on average, against 24 seconds for the hybrid processing. In the 25Kbit bandwidth configuration, this difference is lower, 13 seconds and 11 seconds for the Cloud-centric and Hybrid, respectively. The higher the bandwidth, the lower will be the difference between the two processing approaches. This is because image transmission is the most time-consuming task among the other tasks (*i.e.*, compressing/decompressing images and model inference).



(a) Grid'5000 + FIT IoT LAB. (b) Grid'5000 + FIT IoT LAB.



(c) Chameleon + CHI@Edge. (d) Chameleon + CHI@Edge.

Figure 7.5 – Cloud-centric vs Edge + Cloud processing: (a, b) executed by authors on Grid'5000 and FIT IoT LAB testbeds; and (c, d) replicated by readers on Chameleon Cloud and CHI@Edge.

Amount of data sent to the Cloud

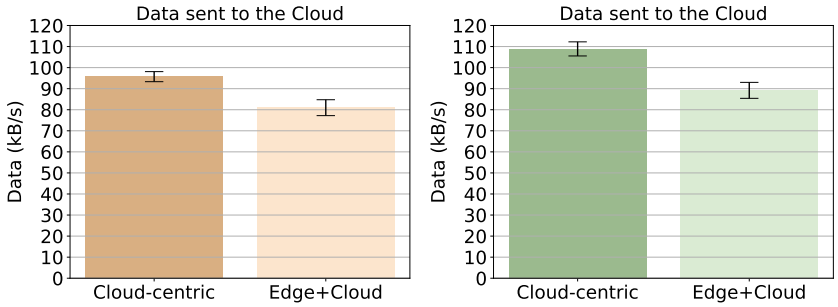
According to the results presented in Figure 7.6a, authors observe that the Hybrid (Edge + Cloud) approach transmits fewer data (81kB/s on average) to the Cloud compared to the Cloud-centric approach (96kB/s on average). This is because, in Hybrid processing, Edge devices compress images before transmitting them to the Cloud.

Resource consumption on the Edge device

Results in Figures 7.7a and 7.7b show that there is no significant difference in the CPU and memory usage in the Edge device when changing between the Cloud-centric and Hybrid processing approaches. CPU usage is around 4.2% and 4.4% for Hybrid and Cloud-centric processing, respectively. Memory usage is around 0.38GB for both.

7.4.3 How KheOps helps readers

After the authors publish their results, other researchers from a different lab download the article from a scientific database and decide to replicate the study on their own premises (*e.g.*, on a different testbed). Following the same logic, we present how KheOps helps the readers



(a) Grid'5000 + FIT IoT LAB. (b) Chameleon + CHI@Edge.

Figure 7.6 – Amount of data sent to the Cloud regarding the Cloud-centric and Edge + Cloud processing approaches.

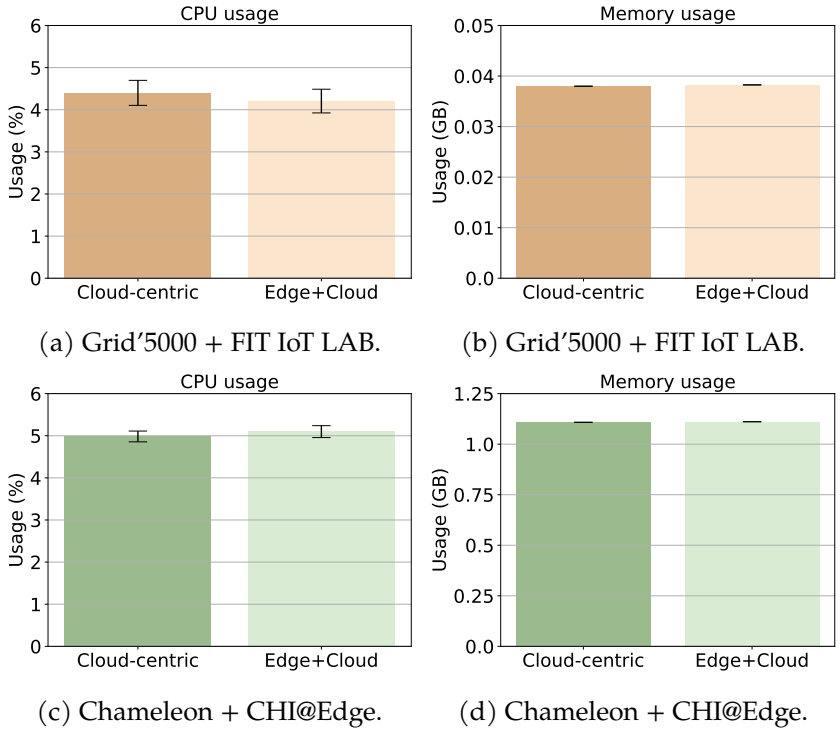
to replicate the experiments cost-effectively, that is, according to the readers requirements (r-REQ).

r-REQ 1. Find and access the experiment as simple as finding and reading the paper Through the KheOps web interface (step 1 in Figure 7.3) the readers obtain access to all the public experiments shared by the community and available in Trovi. Then, they select the experiment shared by the authors of the article to get more details.

r-REQ 2. Perform the experiment, not just read about it Next, in the experiment details web page, readers can launch a JupyterLab server with artifacts in just a single click (steps 2, 3, and 4 in Figure 7.3). Finally, following the experiment instructions described in the Jupyter notebook, the readers deploy and execute the experiments on their testbeds, such as (but not limited to): the Chameleon Cloud and CHI@Edge (step 5 in Figure 7.3).

r-REQ 3. Experiment reasoning: *"What"*, *"Why"*, and *"How"* Before running the experiments, the readers can go through the Jupyter notebook to understand *What* the experiment does (*e.g.,* capture and compress images on Edge devices and then decompress the images and predict the animals on the Cloud server). The readers can also discover *Why* the authors set up the experiment with a 25kbit and 15kbit network bandwidth. Finally, KheOps allows understanding *How* the authors interconnect the Edge devices with the Cloud server (*e.g.,* assigning a public IP to the Cloud server, or opening firewall rules; using the MQTT protocol; *etc.*).

r-REQ 4. Efficiently configure the experimental infrastructure To achieve this, the readers just have to adapt the *layers_services* configuration file (presented in Listing 7.1) to the Chameleon Cloud and CHI@Edge testbeds. Configuring the network bandwidth to 25kbit and then changing it to 15kbit is as simple as changing the *rate* parameter in the *network* file (Listing 7.3). Finally,



(a) Grid'5000 + FIT IoT LAB. (b) Grid'5000 + FIT IoT LAB.

(c) Chameleon + CHI@Edge. (d) Chameleon + CHI@Edge.

Figure 7.7 – Resource consumption on the Edge device: CPU and Memory usage.

copying data to the Edge device, interconnecting it with the Cloud server, launching the application, and finally collecting the results is as simple as defining the *workflow* configuration file (Listing 7.4). The *network* and *workflow* configuration files are testbed agnostic, meaning that users do not need to update these files when changing the deployment from Grid'5000 + FIT IoT LAB to Chameleon Cloud + CHI@Edge.

Next, we report on the replicated experiments carried out by the readers.

Impact of the network on the processing time

From the results in Figures 7.5c and 7.5d, readers conclude that the Hybrid (Edge + Cloud) processing approach outperforms the Cloud-centric one for both network configurations. This conclusion is consistent with the results observed in the published article.

Following the analysis, readers observe that in the 15Kbit bandwidth network configuration, the processing time for the Cloud-centric is about 8 seconds on average, against 6.5 seconds for the hybrid processing. In the 25Kbit bandwidth setup, this difference is lower, 5.5 seconds and 4 seconds for the Cloud-centric and Hybrid, respectively. Similarly to the authors results, readers also observe that the higher the bandwidth, the lower will be the difference between the two processing approaches.

Table 7.2 – Accuracy of replicated experiments.

Metric	Replicability accuracy	Experiment result
Processing time 15Kbit	0.943	Figure 7.5a and 7.5c
Processing time 25Kbit	0.882	Figure 7.5b and 7.5d
Data sent to the cloud	0.973	Figure 7.6a and 7.6b
CPU usage	0.978	Figure 7.7a and 7.7c
Memory usage	0.996	Figure 7.7b and 7.7d

Furthermore, as presented in Table 7.2, we highlight that readers obtained a replicability accuracy of 88.2% and 94.3% for 15Kbit and 25Kbit network configurations, respectively.

Amount of data sent to the Cloud

According to the results presented in Figure 7.6b, readers observe that the Hybrid approach transmits fewer data than the Cloud-centric. The former transmits around 89.2kB/s and the latter 108.8kB/s. Compressing images on the Edge helps to reduce the amount of data sent to the Cloud server. This conclusion is also consistent with the published article and presents a replicability accuracy of 97.3%.

Resource consumption on the Edge device

Results in Figures 7.7c and 7.7d show no significant difference in the CPU and memory usage between the Cloud-centric and the Hybrid processing approaches. CPU usage is around 5.1% and 5% for Hybrid and Cloud-centric processing, respectively. Memory usage is around 1.1GB for both. We highlight that these conclusions are consistent with the published article and present a replicability accuracy of 97.8% and 99.6% for CPU and memory usage, respectively.

7.5 Discussion

We believe that the core idea behind KheOps exhibits several features that make it a promising environment for advancing Computing Continuum research through reproducible and replicable experiments. We briefly discuss them here.

7.5.1 Replicability Accuracy

Despite readers observing a lower processing time compared to the authors, they could verify that their experiment conclusions are consistent with the ones reported by the authors, and their results present a high replicability accuracy (see Table 7.2).

This time difference is expected since readers used a more powerful Edge device (Raspberry Pi4 against Raspberry Pi3) for processing the most time-consuming task (*e.g.,* image compression and then transmission). The Raspberry Pi4 has more RAM memory (8GB *vs.* 1GB in Raspberry Pi3), a better CPU (1.5GHz *vs.* 1.2GHz), network (5GHz *vs.* 2.4GHz).

Furthermore, readers observe slight differences when replicating the original study in different testbeds regarding the remaining metrics, such as the amount of data sent to the Cloud and the CPU and memory usage on the Edge device. This is due to the different deployment approaches used by each testbed. For instance, in FIT IoT LAB, the Raspberry Pi 3 board runs an embedded Linux built with Yocto [238], while CHI@Edge is based on Docker [69] containers. Despite that, the conclusions observed by authors and readers are the same and present high accuracies.

7.5.2 Usability and Reusability

KheOps targets **usability** by allowing users to easily find experiment artifacts shared in Trovi and then launch experiments in a JupyterLab server in just a few clicks. KheOps abstracts all the low-level details of defining and configuring the experimental environment. It provides a high-level abstraction for mapping application parts with the Edge and Cloud infrastructures. Besides, the configuration files used to define the whole experimental environment are designed to be easy to use and understand.

KheOps also targets **reusability** of the experiment artifacts. For instance, readers of an article can reuse the authors artifacts to replicate the study or build upon the existing artifacts to generate new results. In addition, through E2Clab *User-Defined Services*, users can define their own services (*e.g.,* the Edge client and the Cloud server) with the desired deployment logic (*e.g.,* mapping the services to the physical machines/devices; installing required software and packages; *etc.*). The community can share such services in this repository [76]. It allows users to easily plug in the existing services in their experiments. Several services already implemented and shared by the community can be reused, such as Horovod [195], Flower [30], Apache Flink [44], COMPSs [21], among others.

7.5.3 Analyzing other Real-life Applications

The KheOps approach is **generic** in terms of deployment and analysis of **other applications**. Despite our evaluations focusing on the African savanna use-case, we highlight that KheOps can be easily used in other contexts. Supporting new applications can be achieved by describing and implementing their logic in the *User-Defined Services* configuration file.

7.5.4 Integration with other Scientific Testbeds

The KheOps approach is **generic** concerning the **deployment testbeds**. KheOps allows users to analyze application workflows on various large-scale scientific testbeds beyond the four testbeds used in this work. The definition of the experimental environment through E2Clab configuration files (*e.g.*, *layer_services.yaml*, *network.yaml*, and *workflow.yaml*) is testbed agnostic, meaning that deployment on the Grid'5000 testbed can be easily replicated in Chameleon if the latter also provides the required computing resources for the deployment.

7.5.5 Reproducibility and Artifact Availability

The experimental evaluations presented in this work follow a rigorous methodology [176] to support reproducible Edge-to-Cloud experiments on large-scale scientific testbeds. All the experiment artifacts are publicly available [82] at the Trovi sharing portal, and the results are also publicly available [161] in our GitLab repository.

Conclusion

This chapter proposed KheOps. KheOps is, to the best of our knowledge, the first collaborative environment **supporting the cost-effective reproducibility** of applications **on the Edge-to-Cloud Continuum**. It provides simplified abstractions for systematically defining and explaining the experimental environment through **Jupyter notebooks** (*e.g.*, infrastructures, services, network, and workflow execution); explores the **E2Clab methodology** for experimenting on large-scale scientific testbeds providing access to heterogeneous computing resources from the IoT/Edge to the Cloud/HPC; and allows researchers to easily find and share the experiment artifacts in the **Trovi sharing portal**.
The experimental validation shows that KheOps **helps authors** to make their **experiments repeatable and reproducible** on the Grid'5000 and FIT IoT LAB testbeds. Furthermore, KheOps **helps readers** to **cost-effectively replicate authors experiments** in different infrastructures, such as Chameleon Cloud + CHI@Edge testbeds, and obtain the same conclusions with accuracies >88% for all performance metrics.

Chapter 8

Conclusion and Prospects

Contents

Distributed infrastructures for computation and analytics are evolving towards an interconnected ecosystem allowing complex applications to be executed from IoT/Edge devices to the HPC/Cloud. Understanding and optimizing performance in such a complex continuum is challenging. This breaks down to reconciling many, typically contradicting application requirements and constraints with low-level infrastructure design choices. One crucial challenge is accurately reproducing and replicating the relevant behaviors of a given application workflow and representative settings of the physical infrastructure underlying this complex continuum.

Therefore, realizing the Computing Continuum vision in practice raises many questions, which we addressed in this thesis.

1. A first step towards reducing this complexity and enabling the Computing Continuum vision is to enable a **holistic understanding of performance** in such environments. For instance, finding a rigorous approach to answering a question like *Which system parameters and network configurations impact the application performance and how?*
2. A natural next step is to **optimize the performance** of application workflows executed on the highly heterogeneous Computing Continuum. Such applications typically need to comply with various requirements and conflicting constraints, resulting in a vast combination of configuration possibilities and hence complex search space. For instance, it requires answering questions like *Where should the workflow components be executed across the Edge-to-Cloud Continuum to minimize communication costs and end-to-end latency?*

3. **Provenance data capture** may assist the processes of **understanding and optimizing performance** of Edge-to-Cloud workflows. Capturing provenance during workflow execution helps users track inputs, outputs, and processing history, allowing them to steer their experiments precisely. For instance, it helps answer questions like *After multiple workflow evaluations, can we compare their provenance and see what has changed?*
4. Finally, **reproducing** and **replicating** application **performance trade-offs** carried out on large-scale distributed and heterogeneous Edge-to-Cloud infrastructures is non-trivial. These processes compel a lot of effort, are time-consuming, and bring many technical challenges for researchers. One relevant challenge is *How to allow researchers to repeat, reproduce, and replicate Edge-to-Cloud experiments cost-effectively?*

The specific contributions of this thesis toward answering the above questions are detailed in the next section. We then discuss the prospects opened for future work.

8.1 Achievements

8.1.1 Understanding and Optimizing Performance of Edge-to-Cloud Workflows

The E2Clab approach: As a first step toward enabling the Computing Continuum vision, we introduce the E2Clab approach. E2Clab implements a rigorous experiment methodology and is the first framework to support the complete analysis cycle of an application on the Computing Continuum: (*i*) the configuration of the experimental environment, libraries, and frameworks; (*ii*) the mapping between the application parts and machines on the Edge, Fog and Cloud; (*iii*) the deployment of the application on the infrastructure (*e.g.*, Chameleon Cloud, CHI@Edge, Grid'5000, and FIT IoT LAB); (*iv*) the automated execution; and (*v*) the gathering of experiment metrics. E2Clab is open-source [75] and documented [225]. As we finish this manuscript, this implementation reaches version 1.0.0. E2Clab is used in several contributions of this manuscript and cited and used by the research community [174].

Optimization support in E2Clab: First, we use E2Clab to understand the performance of Edge-to-Cloud applications. Next, as such applications are subject to complex constraints and requirements regarding performance, resource usage, and energy consumption, we explore their performance optimization. We propose an optimization methodology and implement it as an extension of E2Clab. It relies on a rigorous analysis of possible application configurations to understand their behavior and related performance trade-offs. We illustrate our methodology by optimizing Pl@ntNet (a worldwide plant identification application) on the Grid'5000 testbed. The results show that E2Clab allows one to understand and improve performance by correlating the parameter settings, the resource usage, and the specifics of the underlying in-

frastructure. Our methodology can be generalized to other applications in the Edge-to-Cloud Continuum.

8.1.2 Enabling Provenance Capture of Edge-to-Cloud Workflows

Capturing the provenance of key performance indicators, with their related data and processes, may assist in understanding and optimizing Edge-to-Cloud workflow executions. However, the provenance capture overhead can be prohibitive, particularly in resource-constrained devices like those on the IoT/Edge. To specifically address this issue, we proposed ProvLight.

The ProvLight approach: ProvLight aims to enable efficient provenance capture on the IoT/Edge. We integrate ProvLight into the E2Clab framework to enable workflow provenance capture across the Edge-to-Cloud Continuum. We validate ProvLight with synthetic workloads on real-life IoT/Edge devices in the large-scale FIT IoT LAB testbed. Evaluations show that ProvLight outperforms state-of-the-art provenance systems. ProvLight is 26—37x faster in capturing and transmitting provenance data, uses 5—7x less CPU, 2x less memory, transmits 2x less data, and consumes 2—2.5x less energy. ProvLight [162] and its integration into E2Clab [75] are available as open-source tools.

8.1.3 Facilitating Reproducibility and Replicability of Edge-to-Cloud Workflows

To understand the performance trade-offs of workflows deployed on the complex Edge-to-Cloud Continuum, one needs to systematically perform experiments to enable their reproducibility and to allow other researchers to replicate the study and the obtained conclusions on different infrastructures. This breaks down to the tedious process of reconciling the numerous experimental requirements and constraints with low-level infrastructure design choices. To this end, we proposed KheOps.

The KheOps approach: KheOps is a collaborative environment designed to enable cost-effective reproducibility and replicability of Edge-to-Cloud experiments. KheOps is composed of three core elements: (1) an experiment repository (*e.g.,* Trovi portal); (2) a notebook environment (*e.g.,* Jupyter); and (3) a multi-platform experiment methodology (*e.g.,* E2Clab). We illustrate KheOps with a real-life Edge-to-Cloud application. The evaluations explore the point of view of the authors of an experiment described in an article (who aim to make their experiments reproducible) and the perspective of their readers (who aim to replicate the experiment). The results show how KheOps helps authors to systematically perform repeatable and reproducible experiments on the Grid5000 + FIT IoT LAB testbeds. Furthermore, KheOps helps readers to cost-effectively replicate authors experiments in different infrastructures, such as Chameleon

Cloud + CHI@Edge testbeds, and obtain the same conclusions with high accuracies (>88% for all performance metrics).

8.2 Prospects

This thesis opens several prospects. Next, we list the most promising ones. We divide these prospects into two sections: the first groups potential contributions related to E2Clab (and ProvLight into E2Clab), while the second lists the directions opened by KheOps.

8.2.1 Prospects Related to the E2Clab (and ProvLight) Approach

Reproducible AI research on the Edge-to-Cloud Continuum: The proper selection of Machine Learning techniques for fast and accurate decision-making requires extensive experiments and evaluations on real-life hybrid infrastructures combining HPC, Cloud, and IoT/Edge systems. The goal is to understand how: (a) infrastructure design choices, (b) optimized learning algorithms with tunable parameters, and (c) the combination of learning paradigms impact performance metrics such as memory usage, energy consumption, model accuracy, training time, network overhead, and application processing latency [173]. This concerns answering questions like (*1*) *How to efficiently deploy AI workflows on heterogeneous Edge-to-Cloud infrastructures to reduce training time and improve model accuracy?* (2) *How to combine Machine Learning paradigms to leverage the massively distributed resources for training across the Edge-to-Cloud Continuum?*

A relevant challenge, worthy of further consideration, is **to understand the performance trade-offs at scale of combining a variety of learning paradigms** such as Reinforcement Learning [228], Online Learning [68], Multi-task Learning [68], and Federated Learning [16]. For instance, exploring the massively distributed IoT/Edge devices for AI training to achieve scalable and distributed deployment of models on Edge-to-Cloud infrastructures; capturing provenance data of model training to compare performance trade-offs; applying Neural Architecture Search [79] and Hyperparameter Search [48] to obtain Deep Learning networks that require less resource without losing accuracy; and exploring Knowledge Distillation [45] (*i.e.*, transferring knowledge from a large model to a smaller model without loss of validity) to leverage model deployment on resource-limited devices.

Lastly, further research is needed on novel approaches proposing rigorous methodologies and systems for **enabling the reproducible performance comparison of AI models and learning paradigms** deployed on large-scale and heterogeneous Edge-to-Cloud infrastructures. Such approaches should support reproducible and replicable experiments. These directions are still ongoing and active research areas in the Big Data and AI communities. As presented in this thesis, we have not seen studies exploring such challenges at a large scale on hybrid Edge-to-Cloud infrastructures.

8.2.2 Prospects Related to the KheOps Approach

Towards a standard platform for facilitating the Reproducibility Initiative in Computer Science: Science has the fundamental idea that progress depends on the ability of independent researchers to verify and reproduce the experimental results using the article artifacts and then to build on them in future studies [171]. Reproducing complex experimental processes demands greater transparency. Providing just experiment descriptions for the reproducibility of the results is not enough and results in a challenging task for the independent researcher. Therefore, transparency in the experimental processes, such as defining the infrastructure configurations, hardware, software, and computer code used to obtain results, is essential for reproducibility.

The Reproducibility Initiative aims to enable reproducible research through transparency and availability of experiment artifacts associated with publications. Various Computer Science conferences and journals, such as SC [191], SIGCOMM [198], and TPDS [217], just to cite a few, are already adopting the Reproducibility Initiative. It consists of authors submitting the artifacts associated with their published article for post-publication peer review. The goal is that reviewers can easily reproduce the article main results and claims with provided artifacts. As an output of this process, articles may receive a badge that quantifies their degree of reproducibility [17].

In this context, reviewers typically find many barriers to reproducing the results, given the large variety of experiment types. For instance, reviewers have to first understand the artifact description, then to have access to the required computing resources (*e.g.,* technologies like GPU, SSD, RDMA, 25G Ethernet, Infiniband, *etc.*), and finally to configure software and execute the experiments [192].

Therefore, a standard platform can be a solution to reduce such complexities. The idea is that authors, through a standard platform, can prepare, execute, and package their experiments so that reviewers and the research community can reproduce them. Ideally, such a platform should be integrated with large-scale scientific testbeds that provide access to a large variety of computing resources, are highly reconfigurable and controllable, and are designed to support reproducible research. Testbeds with such characteristics are explored in this thesis, such as Chameleon, Grid5000, FIT IoT LAB, and CHI@Edge.

BIBLIOGRAPHY

[1] Martín Abadi et al., « Tensorflow: A system for large-scale machine learning », *in*: *12th {USENIX} symposium on operating systems design and implementation ({OSDI} 16)*, 2016, pp. 265–283.

[2] SM Abu Adnan Abir et al., « Iot-enabled smart energy grid: Applications and challenges », *in*: *IEEE access* 9 (2021), pp. 50961–50981.

[3] David Perez Abreu et al., « A Comparative Analysis of Simulators for the Cloud to Fog Continuum », *in*: *Simulation Modelling Practice and Theory* (2019), p. 102029.

[4] Cedric Adjih 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.

[5] *AI Hub.* Jan. 2023, URL: https://aihub.cloud.google.com/.

[6] Brenno M Alencar et al., « FoT-Stream: A Fog platform for data stream analytics in IoT », *in*: *Computer Communications* (2020).

[7] Muhammad Ali et al., « RES: Real-time video stream analytics using edge enhanced clouds », *in*: *IEEE Transactions on Cloud Computing* (2020).

[8] *AMQP: Advanced Message Queuing Protocol.* 2023, URL: http://docs.oasis-open.org/amqp/core/v1.0/os/amqp-core-overview-v1.0-os.html.

[9] Jason Anderson and Kate Keahey, « A case for integrating experimental containers with notebooks », *in*: *2019 IEEE International Conference on Cloud Computing Technology and Science (CloudCom)*, IEEE, 2019, pp. 151–158.

[10] *Ansible Documentation*, URL: https://docs.ansible.com/ansible/latest/index.html, (accessed: 05.03.2020).

[11] *Apache Kafka*, URL: https://kafka.apache.org/, (accessed: 04.27.2020).

[12] *Apache Nifi*, URL: https://nifi.apache.org/, (accessed: 04.27.2020).

[13] *Apache Pulsar*, URL: https://pulsar.apache.org/, (accessed: 04.27.2020).

[14] *Apache Storm*, URL: http://storm.apache.org/, (accessed: 04.27.2020).

[15] AppDynamics, *16 metrics to ensure mobile app success*, Jan. 2015, URL: https://www.appdynamics.com/media/uploaded-files/1432066155/white-paper-16-metrics-every-mobile-team-should-monitor.pdf.

[16] Atakan Aral, Melike Erol-Kantarci, and Ivona Brandić, « Staleness control for edge data analytics », *in*: *Proceedings of the ACM on Measurement and Analysis of Computing Systems* 4.2 (2020), pp. 1–24.

[17] *Artifact Review and Badging Version 1.1.* URL: https://www.acm.org/publications/policies/artifact-review-and-badging-current, (accessed: 22.12.2022).

[18] M Asch 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 », *in*: *The International Journal of High Performance Computing Applications* 32.4 (2018), pp. 435–479.

[19] Mohammad S Aslanpour, Sukhpal Singh Gill, and Adel N Toosi, « Performance evaluation metrics for cloud, fog and edge computing: A review, taxonomy, benchmarks and standards for future research », *in*: *Internet of Things* (2020), p. 100273.

[20] Mehdi Assefi et al., « Big data machine learning using apache spark MLlib », *in*: *2017 ieee international conference on big data (big data)*, IEEE, 2017, pp. 3492–3498.

[21] Rosa M Badia et al., « Comp superscalar, an interoperable programming framework », *in*: *SoftwareX* 3 (2015), pp. 32–36.

[22] Rosa M Badia et al., « Workflow environments for advanced cyberinfrastructure platforms », *in*: *2019 IEEE 39th International Conference on Distributed Computing Systems (ICDCS)*, IEEE, 2019, pp. 1720–1729.

[23] Maximilian Balandat et al., « BoTorch: A Framework for Efficient Monte-Carlo Bayesian Optimization », *in*: *Advances in Neural Information Processing Systems 33*, 2020, URL: http://arxiv.org/abs/1910.06403.

[24] Prasanna Balaprakash et al., « DeepHyper: Asynchronous hyperparameter search for deep neural networks », *in*: *2018 IEEE 25th international conference on high performance computing (HiPC)*, IEEE, 2018, pp. 42–51.

[25] Magda Balazinska et al., « 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.

[26] Ilya Baldin et al., « Exogeni: A multi-domain infrastructure-as-a-service testbed », *in*: *The GENI Book*, Springer, 2016, pp. 279–315.

[27] L. A. Barba and G. K. Thiruvathukal, « Reproducible Research for Computing in Science Engineering », *in*: *Computing in Science Engineering* 19.6 (2017), pp. 85–87.

[28] Khalid Belhajjame et al., « Prov-dm: The prov data model », *in*: *W3C Recommendation* 14 (2013), pp. 15–16.

[29] Julian Bellendorf and Zoltán Ádám Mann, « Classification of optimization problems in fog computing », *in*: *Future Generation Computer Systems* 107 (2020), pp. 158–176.

[30] Daniel J Beutel et al., « Flower: A friendly federated learning research framework », *in*: *arXiv preprint arXiv:2007.14390* (2020).

[31] Raphaël Bolze et al., « Grid'5000: A Large Scale And Highly Reconfigurable Experimental Grid Testbed », *in*: *International Journal of High Performance Computing Applications* 20.4 (2006), pp. 481–494, DOI: 10.1177/1094342006070078, URL: https://hal.inria.fr/hal-00684943.

[32] Raphaël Bolze et al., « Grid'5000: a large scale and highly reconfigurable experimental grid testbed », *in*: *The International Journal of High Performance Computing Applications* 20.4 (2006), pp. 481–494.

[33] Raphaël Bolze et al., « Grid'5000: A Large Scale And Highly Reconfigurable Experimental Grid Testbed », *in*: *International Journal of High Performance Computing Applications* 20.4 (2006), pp. 481–494, DOI: 10.1177/1094342006070078, URL: https://hal.inria.fr/hal-00684943.

[34] Peter A Boncz, Marcin Zukowski, and Niels Nes, « MonetDB/X100: Hyper-Pipelining Query Execution. », *in*: *Cidr*, vol. 5, 2005, pp. 225–237.

[35] Stefan Bouckaert et al., « The w-iLab. t testbed », *in*: *International Conference on Testbeds and Research Infrastructures*, Springer, 2010, pp. 145–154.

[36] Mohamed Amine Bouhlel et al., « A Python surrogate modeling framework with derivatives », *in*: *Advances in Engineering Software* (2019), p. 102662, ISSN: 0965-9978, DOI: https://doi.org/10.1016/j.advengsoft.2019.03.005.

[37] Leo Breiman, « Random forests », *in*: *Machine learning* 45.1 (2001), pp. 5–32.

[38] Eric A Brewer, « Kubernetes and the path to cloud native », *in*: *Proceedings of the sixth ACM symposium on cloud computing*, 2015, pp. 167–167.

[39] Bouziane Brik, Pantelis A Frangoudis, and Adlen Ksentini, « Service-oriented MEC applications placement in a federated edge cloud architecture », *in*: *ICC 2020-2020 IEEE international conference on communications (ICC)*, IEEE, 2020, pp. 1–6.

[40] A. Brogi and S. Forti, « QoS-aware Deployment of IoT Applications Through the Fog », *in*: *IEEE Internet of Things Journal* 4.5 (Oct. 2017), pp. 1185–1192, ISSN: 2327-4662, DOI: 10.1109/JIOT.2017.2701408.

[41] Zhicheng Cai, Qianmu Li, and Xiaoping Li, « Elasticsim: A toolkit for simulating workflows with cloud resource runtime auto-scaling and stochastic task execution times », *in*: *Journal of Grid Computing* 15.2 (2017), pp. 257–272.

[42] Caida, *About Caida*, https://www.caida.org/about/, July 28, 2021.

[43] Rodrigo N Calheiros et al., « CloudSim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms », *in*: *Software: Practice and experience* 41.1 (2011), pp. 23–50.

[44] Paris Carbone et al., « Apache Flink: Stream and Batch Processing in a Single Engine », *in*: *Bulletin of the IEEE Computer Society Technical Committee on Data Engineering* 38.4 (Dec. 2015), pp. 1–11.

[45] Jiasi Chen and Xukan Ran, « Deep Learning With Edge Computing: A Review. », *in*: *Proceedings of the IEEE* 107.8 (2019), pp. 1655–1674.

[46] Yitao Chen et al., « Exploring the use of synthetic gradients for distributed deep learning across cloud and edge resources », *in*: *2nd {USENIX} Workshop on Hot Topics in Edge Computing (HotEdge 19)*, 2019.

[47] Ronan-Alexandre Cherrueau, Matthieu Simonin, and Alexandre Van Kempen, « EnosStack: A LAMP-like Stack for the Experimenter », *in*: *IEEE INFOCOM 2018-IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, IEEE, 2018, pp. 336–341.

[48] Marc Claesen and Bart De Moor, « Hyperparameter search in machine learning », *in*: *arXiv preprint arXiv:1502.02127* (2015).

[49] April Clyburne-Sherin, Xu Fei, and Seth Ariel Green, « Computational reproducibility via containers in psychology », *in*: *Meta-psychology* 3 (2019).

[50] *CoAP: The Constrained Application Protocol.* 2023, URL: https://datatracker.ietf.org/doc/html/rfc7252.

[51] *Code Ocean Explore: Open Science Library.* Jan. 2023, URL: https://codeocean.com/explore.

[52] *Colab: Cloud Storage from the command line.* Jan. 2023, URL: https://cloud.google.com/storage/docs/gsutil.

[53] *Colab: Google Spreadsheets.* Jan. 2023, URL: https://github.com/burnash/gspread#more-examples.

[54] *Compute skylake cluster at CHI@TACC.* Feb. 2023, URL: https://www.chameleoncloud.org/hardware/node/sites/tacc/clusters/chameleon/nodes/0b0bceb9-14bf-423e-890f-3ef187511d71/.

[55] Antonio Coutinho et al., « Fogbed: A rapid-prototyping emulation environment for fog computing », *in*: *2018 IEEE International Conference on Communications (ICC)*, IEEE, 2018, pp. 1–7.

[56] Alexander I Cowen-Rivers et al., « HEBO: Heteroscedastic Evolutionary Bayesian Optimisation », *in*: *arXiv preprint arXiv:2012.03826* (2020), winning submission to the NeurIPS 2020 Black Box Optimisation Challenge.

[57] Geoff Cumming, Fiona Fidler, and David L Vaux, « Error bars in experimental biology », *in*: *The Journal of cell biology* 177.1 (2007), pp. 7–11.

[58] *Dahu cluster.* Feb. 2023, URL: `https://www.grid5000.fr/w/Grenoble:Hardware#dahu`.

[59] Swagatam Das, Sankha Subhra Mullick, and Ponnuthurai N Suganthan, « Recent advances in differential evolution–an updated survey », *in*: *Swarm and Evolutionary Computation* 27 (2016), pp. 1–30.

[60] Rustem Dautov and Salvatore Distefano, « Stream processing on clustered edge devices », *in*: *IEEE Transactions on Cloud Computing* (2020).

[61] Rustem Dautov, Salvatore Distefano, and Rajkumaar Buyya, « Hierarchical data fusion for Smart Healthcare », *in*: *Journal of Big Data* 6.1 (2019), pp. 1–23.

[62] Rustem Dautov et al., « Data processing in cyber-physical-social systems through edge computing », *in*: *IEEE Access* 6 (2018), pp. 29822–29835.

[63] Rustem Dautov et al., « Pushing intelligence to the edge with a stream processing architecture », *in*: *2017 IEEE International Conference on Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData)*, IEEE, 2017, pp. 792–799.

[64] Ewa Deelman et al., « Pegasus, a workflow management system for science automation », *in*: *Future Generation Computer Systems* 46 (2015), pp. 17–35.

[65] Piet Demeester et al., « Fed4fire: the largest federation of testbeds in europe », *in*: *Building the future internet through FIRE*, 2016, pp. 87–109.

[66] Swarnava Dey, Jayeeta Mondal, and Arijit Mukherjee, « Offloaded execution of deep learning inference at edge: Challenges and insights », *in*: *2019 IEEE International Conference on Pervasive Computing and Communications Workshops (PerCom Workshops)*, IEEE, 2019, pp. 855–861.

[67] *DfAnalyzer tool.* Jan. 2018, URL: `https://gitlab.com/ssvitor/dataflow_analyzer/-/tree/master`.

[68] Alberto Diez-Olivan et al., « Data fusion and machine learning for industrial prognosis: Trends and perspectives towards Industry 4.0 », *in*: *Information Fusion* 50 (2019), pp. 92–111.

[69] *Docker.* Jan. 2023, URL: `https://www.docker.com/`.

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

[71] *Docker Hub*, URL: https://hub.docker.com/, (accessed: 05.03.2020).

[72] *Dool (Dstat) monitoring*. Jan. 2018, URL: https://github.com/scottchiefbaker/dool.

[73] Ke-Lin Du and MNS Swamy, « Particle swarm optimization », *in*: *Search and optimization by metaheuristics*, Springer, 2016, pp. 153–173.

[74] *E2Clab experimental artifacts.* May 2021, URL: https://gitlab.inria.fr/E2Clab/Paper-Artifacts/plantnet.

[75] *E2Clab source code.* July 2021, URL: https://gitlab.inria.fr/E2Clab/e2clab.

[76] *E2Clab User Defined Services.* Feb. 2023, URL: https://gitlab.inria.fr/E2Clab/user-defined-services.

[77] *Eclipse Mosquitto*, URL: https://mosquitto.org/, (accessed: 04.27.2020).

[78] *Eclipse Mosquitto: An open source MQTT broker.* Jan. 2017, URL: https://github.com/eclipse/mosquitto.

[79] Thomas Elsken, Jan Hendrik Metzen, and Frank Hutter, « Neural architecture search: A survey », *in*: *The Journal of Machine Learning Research* 20.1 (2019), pp. 1997–2017.

[80] Patricia T Endo et al., « High availability in clouds: systematic review and research challenges », *in*: *Journal of Cloud Computing* 5.1 (2016), pp. 1–15.

[81] ETP4HPC, *ETP4HPC Strategic Research Agenda*, https://www.etp4hpc.eu/sra.html, April 29, 2020.

[82] *Experiment artifacts.* Feb. 2023, URL: https://www.chameleoncloud.org/experiment/share/347adbf3-7c14-4834-b802-b45fdd0d9564.

[83] Xenofon Fafoutis et al., « Extending the battery lifetime of wearable sensors with embedded machine learning », *in*: *2018 IEEE 4th World Forum on Internet of Things (WF-IoT)*, IEEE, 2018, pp. 269–274.

[84] Benedikt Fecher and Sascha Friesike, « Open science: one term, five schools of thought », *in*: *Opening science* (2014), pp. 17–47.

[85] Damián Fernández-Cerero et al., « SCORE: Simulator for cloud optimization of resources and energy consumption », *in*: *Simulation Modelling Practice and Theory* 82 (2018), pp. 160–173.

[86] Marc E Fiuczynski, « Planetlab: overview, history, and future directions », *in*: *ACM SIGOPS Operating Systems Review* 40.1 (2006), pp. 6–10.

[87] *Fog Computing and the Internet of Things: Extend the Cloud to Where the Things Are*, URL: https://www.cisco.com/c/dam/en_us/solutions/trends/iot/docs/computing-overview.pdf, (accessed: 04.21.2020).

[88] Peter I Frazier, « A tutorial on Bayesian optimization », *in*: *arXiv preprint arXiv:1807.02811* (2018).

[89] Jerome H Friedman, « Greedy function approximation: a gradient boosting machine », *in*: *Annals of statistics* (2001), pp. 1189–1232.

[90] Xinwei Fu et al., « Edgewise: a better stream processing engine for the edge », *in*: *2019 {USENIX} Annual Technical Conference ({USENIX}{ATC} 19)*, 2019, pp. 929–946.

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

[92] *G5K hardware: cluster configuration details.* URL: `https://www.grid5000.fr/w/Hardware`, (accessed: 20.12.2022).

[93] Cavide Balkı Gemirter, Çağatay Şenturca, and Şebnem 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.

[94] Ananda Mohon Ghosh and Katarina Grolinger, « Edge-cloud computing for Internet of Things data analytics: Embedding intelligence in the edge with deep learning », *in*: *IEEE Transactions on Industrial Informatics* 17.3 (2020), pp. 2191–2200.

[95] *GitHub.* 2018, URL: `https://github.com/`.

[96] *Google Colab.* Jan. 2023, URL: `https://colab.research.google.com/`.

[97] *Google Colab: Frequently Asked Questions.* Jan. 2023, URL: `https://research.google.com/colaboratory/faq.html`.

[98] *Google Colab vs Kaggle*, Jan. 2023, URL: `https://datasciencenotebook.org/compare/colab/kaggle`.

[99] Maciej Grzenda, Karolina Kwasiborska, and Tomasz Zaremba, « Hybrid short term prediction to address limited timeliness of public transport data streams », *in*: *Neurocomputing* 391 (2020), pp. 305–317.

[100] *GSMA - IoT Edge Computing Requirements*, URL: `https://www.gsma.com/iot/resources/iot-edge-computing-requirements/`, (accessed: 04.23.2020).

[101] Peizhen Guo, Bo Hu, and Wenjun Hu, « Mistify: Automating DNN Model Porting for On-Device Inference at the Edge. », *in*: *NSDI*, 2021, pp. 705–719.

[102] Harshit Gupta et al., « iFogSim: A toolkit for modeling and simulation of resource management techniques in the Internet of Things, Edge and Fog computing environments », *in*: *Software: Practice and Experience* 47.9 (2017), pp. 1275–1296.

[103] Benjamin Haibe-Kains et al., « Transparency and reproducibility in artificial intelligence », *in*: *Nature* 586.*7829* (2020), E14–E16.

[104] DM Hamby, « A comparison of sensitivity analysis techniques », *in*: *Health physics* 68.*2* (1995), pp. 195–204.

[105] Runzhou Han et al., « 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.

[106] Jonathan Hasenburg, Sebastian Werner, and David Bermbach, « Supporting the Evaluation of Fog-based IoT Applications During the Design Phase », *in*: *Proceedings of the 5th Workshop on Middleware and Applications for the Internet of Things* (*M4IoT 2018*), Rennes, France: ACM, 2018.

[107] Manfred Hauswirth and Danh Le-Phuoc, « Autonomous RDF Stream Processing for IoT Edge Devices », *in*: *Semantic Technology: 9th Joint International Conference, JIST 2019, Hangzhou, China, November 25–27, 2019, Proceedings*, vol. 12032, Springer Nature, 2020, p. 304.

[108] Jon C Helton and Freddie Joe Davis, « Latin hypercube sampling and the propagation of uncertainty in analyses of complex systems », *in*: *Reliability Engineering & System Safety* 81.*1* (2003), pp. 23–69.

[109] Melanie Herschel, Ralf Diestelkämper, and Houssem Ben Lahmar, « A survey on provenance: What for? What form? What from? », *in*: *The VLDB Journal* 26 (2017), pp. 881–906.

[110] Rankyung Hong and Abhishek Chandra, « Dlion: Decentralized distributed deep learning in micro-clouds », *in*: *11th {USENIX} Workshop on Hot Topics in Cloud Computing* (*HotCloud 19*), 2019.

[111] Michaela Iorga et al., *Fog Computing Conceptual Model*, tech. rep., 2018.

[112] IoT-LAB, *IoT-LAB A8-M3 board.* `https://www.iot-lab.info/docs/boards/iot-lab-a8-m3/`, January 2, 2023.

[113] Rei Ito, Mineto Tsukada, and Hiroki Matsutani, « An on-device federated learning approach for cooperative model update between edge devices », *in*: *IEEE Access* 9 (2021), pp. 92986–92998.

[114] *Kaggle community.* Jan. 2023, URL: `https://www.kaggle.com/`.

[115] *Kaggle datasets.* Jan. 2023, URL: `https://www.kaggle.com/datasets`.

[116] Kirthevasan Kandasamy et al., « Tuning Hyperparameters without Grad Students: Scalable and Robust Bayesian Optimisation with Dragonfly », *in*: *arXiv preprint arXiv:1903.06694* (2019).

[117] Karamjeet Kaur, Japinder Singh, and Navtej Singh Ghumman, « Mininet as software defined networking testing platform », *in*: *International Conference on Communication, Computing & Systems (ICCCS)*, 2014, pp. 139–42.

[118] Kate Keahey, « The Silver Lining », *in*: *IEEE Internet Computing* 24.4 (2020), pp. 55–59.

[119] Kate Keahey et al., *Chameleon@Edge Community Workshop Report*, 2021.

[120] Kate Keahey et al., « Lessons Learned from the Chameleon Testbed », *in*: *Proceedings of the 2020 USENIX Annual Technical Conference (USENIX ATC '20)*, USENIX Association, July 2020.

[121] Kate Keahey et al., « Lessons learned from the chameleon testbed », *in*: *2020 USENIX Annual Technical Conference (USENIX ATC 20)*, 2020, pp. 219–233.

[122] Staffs Keele et al., *Guidelines for performing systematic literature reviews in software engineering*, tech. rep., Citeseer, 2007.

[123] Mashael Khayyat et al., « Advanced deep learning-based computational offloading for multilevel vehicular edge-cloud computing networks », *in*: *IEEE Access* 8 (2020), pp. 137052–137062.

[124] Ladislav Kocis and William J Whiten, « Computational investigations of low-discrepancy sequences », *in*: *ACM Transactions on Mathematical Software (TOMS)* 23.2 (1997), pp. 266–294.

[125] Ashish Kumar, Saurabh Goyal, and Manik Varma, « Resource-efficient machine learning in 2 KB RAM for the internet of things », *in*: *International Conference on Machine Learning*, PMLR, 2017, pp. 1935–1944.

[126] Dhruv Kumar et al., « Decaf: Iterative collaborative processing over the edge », *in*: *2nd {USENIX} Workshop on Hot Topics in Edge Computing (HotEdge 19)*, 2019.

[127] Rohan Kumar et al., « 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.

[128] Isaac Lera, Carlos Guerrero, and Carlos Juiz, « YAFS: A simulator for IoT scenarios in fog computing », *in*: *IEEE Access* 7 (2019), pp. 91745–91758.

[129] En Li et al., « Edge AI: On-demand accelerating deep neural network inference via edge computing », *in*: *IEEE Transactions on Wireless Communications* 19.1 (2019), pp. 447–457.

[130] Richard Liaw et al., « Tune: A research platform for distributed model selection and training », *in*: *arXiv preprint arXiv:1807.05118* (2018).

[131] Linux, *Linux manual page: tc-netem*, Mar. 2022, URL: `https://man7.org/linux/man-pages/man8/tc-netem.8.html`.

[132] Ling Liu and M Tamer Özsu, *Encyclopedia of database systems*, vol. 6, Springer, 2009.

[133] Yi Liu et al., « Deep anomaly detection for time-series data in industrial IoT: A communication-efficient on-device federated learning approach », *in*: *IEEE Internet of Things Journal* 8.8 (2020), pp. 6348–6358.

[134] Wen-Chih Lo, Chih-Yuan Huang, and Cheng-Hsin 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.

[135] Sidi Lu, Yongtao Yao, and Weisong Shi, « Collaborative learning on the edges: A case study on connected vehicles », *in*: *2nd {USENIX} Workshop on Hot Topics in Edge Computing (HotEdge 19)*, 2019.

[136] Andre Luckow, Kartik Rattan, and Shantenu Jha, « Exploring task placement for edge-to-cloud applications using emulation », *in*: *2021 IEEE 5th International Conference on Fog and Edge Computing (ICFEC)*, IEEE, 2021, pp. 79–83.

[137] Zaigham Mahmood, *Fog Computing: Concepts, Frameworks and Technologies*, Springer, 2018.

[138] Asad Waqar Malik et al., « XFogSim: A distributed fog resource management framework for sustainable IoT services », *in*: *IEEE Transactions on Sustainable Computing* 6.4 (2020), pp. 691–702.

[139] Ruben Mayer et al., « Emufog: Extensible and scalable emulation of large-scale fog computing infrastructures », *in*: *2017 IEEE Fog World Congress (FWC)*, IEEE, 2017, pp. 1–6.

[140] Alberto Medina et al., « BRITE: An approach to universal topology generation », *in*: *MASCOTS 2001, Proceedings Ninth International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems*, IEEE, 2001, pp. 346–353.

[141] Francisco Ferreira de Mendonça Junior et al., « The trade-offs between Fog Processing and Communications in latency-sensitive Vehicular Fog Computing », *in*: *Pervasive and Mobile Computing* 84 (2022), p. 101638.

[142] Sadip Midya et al., « Multi-objective optimization technique for resource allocation and task scheduling in vehicular cloud architecture: A hybrid adaptive nature inspired approach », *in*: *Journal of Network and Computer Applications* 103 (2018), pp. 58–84.

[143] Seyedali Mirjalili, « Genetic algorithm », *in*: *Evolutionary algorithms and neural networks*, Springer, 2019, pp. 43–55.

[144] Roberto 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.

[145] *MQTT: The Standard for IoT Messaging*. 2023, URL: `https://mqtt.org/`.

[146] Dariusz Mrozek, Anna Koczur, and Bożena Małysiak-Mrozek, « Fall detection in older adults with mobile IoT devices and machine learning in the cloud and on the edge », *in*: *Information Sciences* 537 (2020), pp. 132–147.

[147] Lekha R Nair, Sujala D Shetty, and Siddhanth D Shetty, « Applying spark based machine learning model on streaming big data for health status prediction », *in*: *Computers & Electrical Engineering* 65 (2018), pp. 393–399.

[148] Devki Nandan Jha et al., « IoTSim-Edge: A Simulation Framework for Modeling the Behaviour of IoT and Edge Computing Environments », *in*: *arXiv e-prints* (2019), arXiv–1910.

[149] Lucas Nussbaum, « An overview of Fed4FIRE testbeds–and beyond? », *in*: *GEFI-Global Experimentation for Future Internet Workshop*, 2019.

[150] *Open Science*, URL: `https://ec.europa.eu/research/openscience`, (accessed: 05.03.2020).

[151] PROJETO ORBIT, *Open-Access Research Testbed for Next-Generation Wireless Networks*, 2016.

[152] Eva Ostertagová, « Modelling using polynomial regression », *in*: *Procedia Engineering* 48 (2012), pp. 500–506.

[153] Thomas Pasquier et al., « Runtime analysis of whole-system provenance », *in*: *Proceedings of the 2018 ACM SIGSAC conference on computer and communications security*, 2018, pp. 1601–1616.

[154] Adam Paszke et al., « Pytorch: An imperative style, high-performance deep learning library », *in*: *Advances in neural information processing systems* 32 (2019), pp. 8026–8037.

[155] Katerina Pechlivanidou et al., « NITOS testbed: A cloud based wireless experimentation facility », *in*: *2014 26th International Teletraffic Congress (ITC)*, IEEE, 2014, pp. 1–6.

[156] Fabian Pedregosa et al., « Scikit-learn: Machine learning in Python », *in*: *the Journal of machine Learning research* 12 (2011), pp. 2825–2830.

[157] Juan Luis Pérez et al., « A resilient and distributed near real-time traffic forecasting application for Fog computing environments », *in*: *Future Generation Computer Systems* 87 (2018), pp. 198–212.

[158] Manuel Peuster, Johannes Kampmeyer, and Holger Karl, « Containernet 2.0: A rapid prototyping platform for hybrid service function chains », *in*: *2018 4th IEEE Conference on Network Softwarization and Workshops (NetSoft)*, IEEE, 2018, pp. 335–337.

[159] Duc Pham and Dervis Karaboga, *Intelligent optimisation techniques: genetic algorithms, tabu search, simulated annealing and neural networks*, Springer Science & Business Media, 2012.

[160] Débora Pina et al., « 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.

[161] *Provenance Capture IoT/Edge: experiment artifacts.* Jan. 2023, URL: https://gitlab.inria.fr/E2Clab/examples/provenance-iot-edge.

[162] *ProvLight tool: Provenance Capture for IoT/Edge devices.* Jan. 2023, URL: https://gitlab.inria.fr/provlight/provlight.

[163] *Python Client for MQTT-SN brokers.* Jan. 2017, URL: https://github.com/luanguimaraesla/mqttsn.

[164] *Python zlib.* Feb. 2023, URL: https://docs.python.org/3/library/zlib.html.

[165] Tariq Qayyum et al., « FogNetSim++: A toolkit for modeling and simulation of distributed fog environment », *in*: *IEEE Access* 6 (2018), pp. 63570–63583.

[166] RADICAL-DREAMER, *RADICAL-DREAMER: Dynamic Runtime and Execution Adaptive Middleware EmulatoR (RD)*, https://github.com/radical-project/radical.dreamer/, Feb 12, 2022.

[167] Rajiv Ranjan et al., « The next grand challenges: Integrating the internet of things and data science », *in*: *IEEE Cloud Computing* 5.3 (2018), pp. 12–26.

[168] *Raspberry Pi 3 Model B.* Feb. 2023, URL: https://www.iot-lab.info/docs/boards/raspberry-pi-3/.

[169] *Raspberry Pi 4.* Feb. 2023, URL: https://chameleoncloud.org/experiment/chiedge/hardware-info/.

[170] Ray, *What is Ray?*, May 2021, URL: https://docs.ray.io/en/latest/index.html.

[171] *Reporting standards and availability of data, materials, code and protocols*, Feb. 2020, URL: https://www.nature.com/nature-portfolio/editorial-policies/reporting-standards.

[172] *Requests: HTTP for Humans.* Jan. 2018, URL: https://github.com/psf/requests.

[173] Aluizio F Rocha Neto et al., « Distributed machine learning for iot applications in the fog », *in*: *Fog Computing: Theory and Practice* (2020), pp. 309–345.

[174] Daniel Rosendo, *E2Clab Documentation Web Page: citations and usage by the research community.* https://e2clab.gitlabpages.inria.fr/e2clab/publications.html, 2023.

[175] Daniel Rosendo et al., « Distributed intelligence on the Edge-to-Cloud Continuum: A systematic literature review », *in*: *Journal of Parallel and Distributed Computing* 166 (Aug. 2022), pp. 71–94, DOI: 10.1016/j.jpdc.2022.04.004, URL: https://hal.archives-ouvertes.fr/hal-03654722.

[176] Daniel Rosendo et al., « 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, Sept. 2020, pp. 1–11, DOI: 10.1109/CLUSTER49012.2020.00028, URL: https://hal.archives-ouvertes.fr/hal-02916032.

[177] Daniel Rosendo et al., « E2Clab: Exploring the Computing Continuum through Repeatable, Replicable and Reproducible Edge-to-Cloud Experiments », *in*: *2020 IEEE International Conference on Cluster Computing (CLUSTER)*, IEEE, 2020, pp. 176–186.

[178] Daniel Rosendo et al., *E2Clab: Reproducible Analysis of Complex Workflows on the Edge-to-Cloud Continuum*, IPDPS 2021 - 35th IEEE International Parallel and Distributed Processing Symposium, Poster, May 2021, URL: https://hal.archives-ouvertes.fr/hal-03269852.

[179] Daniel Rosendo et al., « KheOps: Cost-effective Repeatability, Reproducibility, and Replicability of Edge-to-Cloud Experiments », *in*: *Proceedings of the 2023 ACM Conference on Reproducibility and Replicability*, 2023, pp. 62–73.

[180] Daniel Rosendo et al., « ProvLight: Efficient Workflow Provenance Capture on the Edge-to-Cloud Continuum », *in*: *Cluster 2023 - IEEE International Conference on Cluster Computing*, Santa Fe, New Mexico, United States, Oct. 2023, In press, URL: https://hal.science/hal-04161546.

[181] Daniel Rosendo et al., « 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, Sept. 2021, pp. 23–34, DOI: 10.1109/Cluster48925.2021.00043, URL: https://hal.archives-ouvertes.fr/hal-03310540.

[182] Daniel Rosendo et al., « Reproducible Performance Optimization of Complex Applications on the Edge-to-Cloud Continuum », *in*: *arXiv preprint arXiv:2108.04033* (2021).

[183] *RPL: IPv6 Routing Protocol for Low-Power and Lossy Networks.* 2023, URL: https://datatracker.ietf.org/doc/html/rfc6550.

[184] *RSMB: Really Small Message Broker.* Jan. 2013, URL: https://github.com/eclipse/mosquitto.rsmb.

[185] Pingcheng Ruan et al., « LineageChain: a fine-grained, secure and efficient data provenance system for blockchains », *in*: *The VLDB Journal* 30.1 (2021), pp. 3–24.

[186] Shazia Sadiq et al., « Data Flow and Validation in Workflow Modelling », *in*: *Proceedings of the 15th Australasian database conference-Volume 27*, 2004, pp. 207–214.

[187] Luis Sanchez et al., « SmartSantander: IoT experimentation over a smart city testbed », *in*: *Computer Networks* 61 (2014), pp. 217–238.

[188] Suresh Sankaranarayanan et al., « Data Flow and Distributed Deep Neural Network based low latency IoT-Edge computation model for big data environment », *in*: *Engineering Applications of Artificial Intelligence* 94 (2020), p. 103785.

[189] David Sarabia-Jácome et al., « Efficient Deployment of Predictive Analytics in Edge Gateways: Fall Detection Scenario », *in*: *2019 IEEE 5th World Forum on Internet of Things (WF-IoT)*, IEEE, 2019, pp. 41–46.

[190] David Sarabia-Jácome et al., « Highly-efficient fog-based deep learning AAL fall detection system », *in*: *Internet of Things* 11 (2020), p. 100185.

[191] *SC Reproducibility Initiative*, Feb. 2023, URL: https://sc23.supercomputing.org/program/papers/reproducibility-initiative/.

[192] *SC: The largest Reproducibility Laboratory*, Feb. 2023, URL: https://www.chameleoncloud.org/blog/2023/02/20/sc-the-largest-reproducibility-laboratory/.

[193] National Academies of Sciences Engineering, Medicine, et al., *Reproducibility and replicability in science*, National Academies Press, 2019.

[194] Scikit-Optimize, *Sequential model-based optimization*, Mar. 2020, URL: https://scikit-optimize.github.io/stable/.

[195] Alexander Sergeev and Mike Del Balso, « Horovod: fast and easy distributed deep learning in TensorFlow », *in*: *arXiv preprint arXiv:1802.05799* (2018).

[196] Shree Krishna Sharma and Xianbin Wang, « Live data analytics with collaborative edge and cloud processing in wireless IoT networks », *in*: *IEEE Access* 5 (2017), pp. 4621–4635.

[197] Christian Sicari et al., « OpenWolf: A Serverless Workflow Engine for Native Cloud-Edge Continuum », *in*: *2022 IEEE Intl Conf on Dependable, Autonomic and Secure Computing, Intl Conf on Pervasive Intelligence and Computing, Intl Conf on Cloud and Big Data Computing, Intl Conf on Cyber Science and Technology Congress (DASC/PiCom/CBDCom/CyberSciTech)*, IEEE, 2022, pp. 1–8.

[198] *SIGCOMM Reproducibility Initiative*, Feb. 2023, URL: https://conferences.sigcomm.org/sigcomm/2023/cfp.html.

[199] Pedro Silva, Alexandru Costan, and Gabriel Antoniu, « Investigating Edge vs. Cloud Computing Trade-offs for Stream Processing », *in*: *IEEE International Conference on Big Data in 2019*, 2019, pp. 469–474.

[200] Rafael Ferreira da Silva et al., « Workflows Community Summit 2022: A Roadmap Revolution », *in*: *arXiv preprint arXiv:2304.00019* (2023).

[201] Rômulo M Silva et al., « 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.

[202] Vítor Silva et al., « Dfanalyzer: runtime dataflow analysis tool for computational science and engineering applications », *in*: *SoftwareX* 12 (2020), p. 100592.

[203] Timothy W Simpson et al., « Kriging models for global approximation in simulation-based multidisciplinary design optimization », *in*: *AIAA journal* 39.*12* (2001), pp. 2233–2241.

[204] Jasper Snoek, Hugo Larochelle, and Ryan P Adams, « Practical bayesian optimization of machine learning algorithms », *in*: *arXiv preprint arXiv:1206.2944* (2012).

[205] Cagatay Sonmez, Atay Ozgovde, and Cem Ersoy, « Edgecloudsim: An environment for performance evaluation of edge computing systems », *in*: *Transactions on Emerging Telecommunications Technologies* 29.*11* (2018), e3493.

[206] Renan Souza et al., « Efficient runtime capture of multiworkflow data using provenance », *in*: *2019 15th International Conference on eScience* (*eScience*), IEEE, 2019, pp. 359–368.

[207] Renan Souza et al., « Keeping Track of User Steering Actions in Dynamic Workflows », *in*: *Future Generation Computer Systems* 99 (2019), pp. 624–643, ISSN: 0167-739X, DOI: 10.1016/j.future.2019.05.011.

[208] Renan Souza et al., « Workflow Provenance in the Lifecycle of Scientific Machine Learning », *in*: *Concurrency and Computation: Practice and Experience* e6544 (2021), pp. 1–21.

[209] S. Sreerangaraju, *Emulation vs. Simulation*, https://www.perfecto.io/blog/emulation-vs-simulation, 2020.

[210] Andy Stanford-Clark and Hong Linh Truong, « Mqtt for sensor networks (mqtt-sn) protocol specification », *in*: *International business machines* (*IBM*) *Corporation version* 1.2 (2013), pp. 1–28.

[211] Ingo Steinwart and Andreas Christmann, *Support vector machines*, Springer Science & Business Media, 2008.

[212] Victoria Stodden and Sheila Miguez, « Best Practices for Computational Science: Software Infrastructure and Environments for Reproducible and Extensible Research », *in*: *Available at SSRN 2322276* (2013).

[213] Jakob Struye et al., « The CityLab testbed—Large-scale multi-technology wireless experimentation in a city environment: Neural network-based interference prediction in a smart city », *in*: *IEEE INFOCOM 2018-IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, IEEE, 2018, pp. 529–534.

[214] Sergej Svorobej et al., « Simulating fog and edge computing scenarios: An overview and research challenges », *in*: *Future Internet* 11.*3* (2019), p. 55.

[215] Ryan Tanaka 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.

[216] Yucel Tas, Mohamed Jehad Baeth, and Mehmet 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.

[217] *TPDS Reproducibility Initiative*, URL: `https://www.computer.org/csdl/journal/td/write-for-us/104303`.

[218] *UDP: User Datagram Protocol.* 2023, URL: `https://datatracker.ietf.org/doc/html/rfc768`.

[219] Peter JM Van Laarhoven and Emile HL Aarts, « Simulated annealing », *in*: *Simulated annealing: Theory and applications*, Springer, 1987, pp. 7–15.

[220] Thomas Vanhove et al., « Tengu: An experimentation platform for big data applications », *in*: *2015 IEEE 35th International Conference on Distributed Computing Systems Workshops*, IEEE, 2015, pp. 42–47.

[221] Parul Verma and Shahnaz Fatima, « Smart Healthcare Applications and Real-Time Analytics Through Edge Computing », *in*: *Internet of Things Use Cases for the Healthcare Industry*, Springer, 2020, pp. 241–270.

[222] Dan Wang et al., « From IoT to 5G I-IoT: The next generation IoT-based intelligent algorithms and 5G technologies », *in*: *IEEE Communications Magazine* 56.*10* (2018), pp. 114–120.

[223] Juan Wang and Di Li, « Adaptive computing optimization in software-defined network-based industrial Internet of Things with fog computing », *in*: *Sensors* 18.*8* (2018), p. 2509.

[224] Xizhao Wang et al., « On the optimization of fuzzy decision trees », *in*: *Fuzzy Sets and Systems* 112.*1* (2000), pp. 117–125.

[225] *Welcome to E2Clab's documentation!*, Feb. 2020, URL: https://e2clab.gitlabpages.inria.fr/e2clab.

[226] Philip Wette et al., « Maxinet: Distributed emulation of software-defined networks », *in*: *2014 IFIP Networking Conference*, IEEE, 2014, pp. 1–9.

[227] *What is Docker Hub?*, URL: https://www.docker.com/products/docker-hub/.

[228] Marco A Wiering and Martijn Van Otterlo, « Reinforcement learning », *in*: *Adaptation, learning, and optimization* 12.3 (2012).

[229] Mark D Wilkinson et al., « The FAIR Guiding Principles for scientific data management and stewardship », *in*: *Scientific data* 3.1 (2016), pp. 1–9.

[230] Bharati Wukkadada et al., « 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.

[231] Ye Xia et al., « 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.

[232] Guangming Xian, « Parallel machine learning algorithm using fine-grained-mode spark on a mesos big data cloud computing software framework for mobile robotic intelligent fault recognition », *in*: *IEEE Access* 8 (2020), pp. 131885–131900.

[233] Ying Xiong et al., « Extend cloud to edge with KubeEdge », *in*: *2018 IEEE/ACM Symposium on Edge Computing (SEC)*, IEEE, 2018, pp. 373–377.

[234] Hansong Xu et al., « A survey on industrial Internet of Things: A cyber-physical systems perspective », *in*: *IEEE Access* 6 (2018), pp. 78238–78259.

[235] Jia Xu et al., « EdgeWorkflow: One click to test and deploy your workflow applications to the edge », *in*: *Journal of Systems and Software* 193 (2022), p. 111456.

[236] Le Xu et al., « Move Fast and Meet Deadlines: Fine-grained Real-time Stream Processing with Cameo », *in*: *18th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 21)*, 2021, pp. 389–405.

[237] *YAML*, URL: https://yaml.org/, (accessed: 05.03.2020).

[238] *Yocto Project.* Jan. 2023, URL: https://www.yoctoproject.org/.

[239] Matei Zaharia et al., « Spark: Cluster Computing with Working Sets », *in*: *Proceedings of the 2Nd USENIX Conference on Hot Topics in Cloud Computing*, HotCloud'10, Boston, MA: USENIX Association, 2010, pp. 10–10.

[240] *Zenodo.* 2018, URL: https://zenodo.org/.

[241] *ZeroMQ*, URL: https://zeromq.org/, (accessed: 04.27.2020).

[242] Tuo Zhang et al., « Federated learning for internet of things », *in*: *Proceedings of the 19th ACM Conference on Embedded Networked Sensor Systems*, 2021, pp. 413–419.

[243] Xingzhou Zhang, Yifan Wang, and Weisong Shi, « pcamp: Performance comparison of machine learning packages on the edges », *in*: {*USENIX*} *Workshop on Hot Topics in Edge Computing* (*HotEdge 18*), 2018.

[244] Li Zhou et al., « Distributing deep neural networks with containerized partitions at the edge », *in*: *2nd* {*USENIX*} *Workshop on Hot Topics in Edge Computing* (*HotEdge 19*), 2019.

[245] Zhe Zhou et al., « SaFace: Towards Scenario-aware Face Recognition via Edge Computing System », *in*: *3rd* {*USENIX*} *Workshop on Hot Topics in Edge Computing* (*HotEdge 20*), 2020.

[246] *Zooniverse dataset*. Feb. 2023, URL: https://www.zooniverse.org/organizations/meredithspalmer/snapshot-safari.

COLLEGE DOCTORAL BRETAGNE
MATHS, TELECOMS
INFORMATIQUE, SIGNAL
SYSTEMES, ELECTRONIQUE

INSA INSTITUT NATIONAL DES SCIENCES APPLIQUÉES RENNES

Titre : Méthodologies pour l'Analyse Reproductible des Workflows Edge-to-Cloud

Mot clés : Continuum Edge-to-Cloud, Reproductibilité, Méthodologie, Workflows

Résumé : Les infrastructures distribuées pour le calcul et l'analyse évoluent désormais vers un écosystème interconnecté permettant l'exécution d'applications complexes dans le continuum Edge-to-Cloud. Comprendre et optimiser les performances de bout en bout dans ce contexte est un défi majeur. Un besoin crucial consiste à reproduire avec précision les comportements pertinents des workflows et les paramètres représentatifs de l'infrastructure physique sous-jascente.

Cette thèse est une contribution conceptuelle et pratique au *continuum Edge-to-Cloud*, proposant des méthodologies et les appliquant dans des environnements novateurs. Nos méthodologies ont pour but de s'affranchir de la complexité de la compréhension et de l'optimisation des workflows dans le continuum Edge-to-Cloud. Ainsi, elles permettent la conception d'expériences reproductibles, l'optimisation des applications, la capture efficace des données de provenance des exécutions de workflows, et la reproductibilité des expériences.

Nous avons validé notre proposition avec, d'abord, le développement du framework E2Clab qui supporte le cycle complet d'analyse d'une application dans le continuum Edge-to-Cloud, et ensuite, l'utilisation de E2Clab pour l'optimisation Pl@ntNet, une application mondiale d'identification des plantes. La validation expérimentale à grande échelle sur Grid'5000 montre que notre méthodologie s'est avérée utile pour comprendre et améliorer les performances de Pl@ntNet.

Title: Methodologies for Reproducible Analysis of Workflows on the Edge-to-Cloud Continuum

Keywords: Edge-to-Cloud Continuum, Reproducibility, Methodology, Workflows

Abstract: Distributed infrastructures for computation and analytics are now evolving towards an interconnected ecosystem allowing complex applications to be executed on the Edge-to-Cloud continuum. Understanding and optimizing end-to-end performance in such a complex continuum is challenging. One crucial challenge is accurately reproducing relevant behaviors of a given application workflow and representative settings of the physical infrastructure underlying this complex continuum.

This thesis is a conceptual and practical contribution to the Edge-to-Cloud continuum, proposing and applying methodologies in novel environments. Our methodologies aim at overcoming the complexity of understanding and optimizing Edge-to-Cloud workflows. As such, they enable reproducible experiment design, application optimization, efficient workflow provenance capture, and cost-effective experiment reproducibility.

We validated our proposal by first developing the E2Clab framework that supports the complete analysis cycle of an application on the Edge-to-Cloud Continuum and then using E2Clab to optimize Pl@ntNet, a global plant identification application. Large-scale experimental validation on Grid'5000 shows that our methodology has proven helpful for understanding and improving the performance of Pl@ntNet.