SPOKE 1
FUTURE HPC & BIG DATA

FLAGSHIP 3:
Selection of candidate prototypes frameworks and development tools for HPC

Finanziato dall'Unione europea NextGenerationEU
Ministero dell'Università e della Ricerca
Italiadomani PIANO NAZIONALE DI RIPRESA E RESILIENZA
ICSC Centro Nazionale di Ricerca in HPC, Big Data and Quantum Computing

EXECUTIVE SUMMARY

This is the second deliverable of Spoke 1 Flagship 3. After presenting the general map of competences of the flagship affiliates in the first deliverable “Survey_of_state-of-the-art_approaches_and_gap_analysis_on_HPC_dev_tools_D4.3”, this deliverable focus on providing an overview of the candidate prototypes that will be the focus of the flagship research and implementation activities until the completion of the project. The candidate prototypes are designed to leverage the synergies among various tools developed by the flagship affiliates, aiming to offer HPC application developers improved tools, methodologies, and technologies. All of this will be incorporated within the context of the cloud/HPC convergence scenario outlined in the flagship description.
After anintroduction, also aimed at correctly placing the prototypes in the overall Cloud/HPC tool stack, Section 2 introduces the different prototypes planned. For each candidate prototype we shortly outline the main features and the final goal, and then we outline its status, the expected improvements, the main cooperations and final validation tests to be used to assess the quality of our work.
In some cases, the candidate prototypes will eventually be applications or use cases demonstrating combined & synergic use of tools developed by different research groups contributing to flagship activities. In other cases, the candidate prototypes will eventually demonstrate the possibility to use some tools in different contexts improving different non-functional features of interest (performance, power consumption, efficiency, …) in either small applications or in other tools.
In all cases, we delineate anticipated outcomes along with methods for evaluating the achieved results.

1 Introduction.....5

2 Candidate prototypes.....9

2.1 Compression of peta-scale collections of textual and source-code files.....9
2.1.1 Introduction.....9
2.1.2 Related works.....9
2.1.3 Actual prototype description and maturity level.....10
2.1.4 Prototype evolution and implementation.....12
2.1.5 Final validation tests.....16

2.2 Astrophysics data analysis and visualization.....16
2.2.1 Introduction.....16
2.2.2 Related works.....17
2.2.3 Actual prototype description and maturity level.....18
2.2.4 Prototype evolution and implementation.....20
2.2.5 Final validation tests.....23

2.3 Genomic variant calling pipeline.....24
2.3.1 Introduction.....24
2.3.2 Related works.....25
2.3.3 Actual prototype description and maturity level.....25
2.3.4 Prototype evolution and implementation.....29
2.3.5 Final validation tests.....30

2.4 Edge-Cloud continuum federation infrastructure.....30
2.4.1 Introduction.....30
2.4.2 Related works.....32
2.4.3 Actual prototype description and maturity level.....33
2.4.4 Prototype evolution and implementation.....44
2.4.5 Final validation tests.....46

2.5 Interactive Computing Service.....47
2.5.1 Introduction.....47
2.5.2 Related works.....47
2.5.3 Actual prototype description and maturity level.....48
2.5.4 Prototype evolution and implementation.....49
2.5.5 Final validation tests.....52

2.6 Serverledge: QoS-Aware Function-as-a-Service in the Edge-Cloud Continuum.....53
2.6.1 Introduction.....53
2.6.2 Related works.....53
2.6.3 Actual prototype description and maturity level.....54
2.6.4 Prototype evolution and implementation.....58
2.6.5 Final validation tests.....60

2.7 Improving I/O phases in computational modelling of Galaxy Formation.....60
2.7.1 Introduction.....60
2.7.2 Related works.....61
2.7.3 Actual prototype description and maturity level.....61
2.7.4 Prototype evolution and implementation.....65
2.7.5 Final validation tests.....65

2.8 WorldDynamics.jl.....66
2.8.1 Introduction.....66
2.8.2 Related works.....66
2.8.3 Actual prototype description and maturity level.....67
2.8.4 Prototype evolution and implementationCompleto.....68
2.8.5 Final validation tests.....70

2.9 Optimized deployment of cloud-native applications over multi-cloud and cloud continuum scenarios.....70
2.9.1 Introduction.....70
2.9.2 Related works.....70

2.9.3 Actual prototype description and maturity level....71
2.9.4 Prototype evolution and implementation....73
2.9.5 Final validation tests....74

2.10 FastFlow: an alternative programming model for HPC applications....74
2.10.1 Introduction....74
2.10.2 Related works....74
2.10.3 Actual prototype description and maturity level....75
2.10.4 Prototype evolution and implementation....78
2.10.5 Final validation tests....79

2.11 Anomalous subgroup characterization with DivExplorer....79
2.11.1 Introduction....79
2.11.2 Related works....80
2.11.3 Actual prototype description and maturity level....80
2.11.4 Prototype evolution and implementation....81
2.11.5 Final validation tests....83

2.12 Compilation flow and deployment strategy targeting RISC-V accelerators for HPC computing 83
2.12.1 Introduction....83
2.12.2 Related works....84
2.12.3 Actual prototype description and maturity level....85
2.12.4 Prototype evolution and implementation....86
2.12.5 Final validation tests....86

2.13 National Federated Cloud/HPC Infrastructure....86
2.13.1 Introduction....87
2.13.2 Related works....87
2.13.3 Actual prototype description and maturity level....87
2.13.4 Prototype evolution and implementation....89
2.13.5 Final validation tests....90

3 References....91

1 Introduction

Representative prototypes, models, and frameworks are essential tools in engineering and science for the development, evaluation, validation, and test of ideas, concepts, and processes [126], [127]. Moreover, these tools provide support and feedback when analyzing new solutions, methods, approaches, schemes, or architectures. In fact, prototypes reduce the gap between first stages of an idea and the realistic implementations by supporting analyses of a system under clear evaluation targets (e.g., evaluate the performance of a multi-processor under specific operational conditions).

In general, most prototypes provide a set of controlled conditions to evaluate the technical feasibility of theories in environments and scenarios that are like the real operation of a system. The quality and functionality of a prototype is given by its fidelity, so high-fidelity models and prototypes include most of the expected functionalities and details of a real system. Similarly, low-fidelity prototypes include a reduced number of features but still allow the evaluation and analysis under limited scenarios. In some cases, one single high-fidelity prototype involves all possible operative scenarios and can be exploited for evaluation and analysis.

On the other hand, one or more high/medium-fidelity and complementary prototypes (i.e., with different abstraction levels) might contribute to evaluate different features and provide consistency, high accuracy, or more extended analyses than one single prototype. Similarly, the use of prototypes can be used to anticipate behaviors in a system and identify unexpected constraints (e.g., physical, technical, or financial) before reaching production phases. Thus, the identification and selection of feasible prototypes and models is a crucial step for the evaluation of several features also in the HPC domain. In fact, one or more prototypes are vital tools to evaluate solutions and management mechanisms against non-functional properties in HPC machines, such as the power management and the monitoring of energy or reliability features.

This delivery report analyzes and selects a set of prototype candidates, frameworks, tools, and artifacts to test and validate if a profitable convergence between HPC and Cloud/edge computing techniques, architectures and tools is possible, to realize what is called the Hybrid Cloud-HPC architecture. Such approach has a certain number of benefits, as described in the introduction to the deliverable D4.FL3, first the possibility to exploit the flexibility in the management and autonomic orchestration of resources, the reliability and the geographical pervasive distribution of the Cloud/edge datacenters and devices, enabling the possibility to execute HPC applications on existed IaaS or FaaS infrastructures. The goal of the D4.FL3 document is to identify the most promising areas for future research and development in HPC development tools, analyzing the current state-of-the-art approaches and identifying gaps.

The HPC architecture defined in the D4.FL3 deliverable will be fully exploited in this deliverable, where we will outline a selection of candidate prototypes exploiting the HPC tools made available by the partners, aimed at showcasing the activities of the partners in the Flagship. In the subsequent deliverables, the candidate prototypes will be extended, integrated, and assessed to demonstrate the advances achieved with respect to the state of the art at the beginning of this project.

Choosing and selecting the proper set of candidate prototypes is challenging, since every partner has different competencies and their tools, described in the D4.FL3 deliverable, cover different levels of the Hybrid Cloud-HPC architecture shown in figure 1. This architecture spans multiple technological stacks and target platforms and, more precisely, is composed of six layers, whose have been deeply analyzed in D4.FL3, starting from the high-level application model layer, down to the runtime application management and hardware optimization layers: such a complex and vertical architecture lead to a great variety of tools, frameworks, and prototypes. In this introduction, we give an idea of the main selection criteria we have decided to apply to select the candidate prototypes that will be described in the section 2 of this document.

The first and main criterion we adopt to choose the candidate prototypes for this document is their capacity to cover the most part of the architecture. The idea is to select prototypes for which an increase in such coverage is possible respect to their actual implementation. For this reason, the second criterion, which is directly connected to the first, is the possibility to integrate at least two tools of different partners in developing the final version of the prototype, with the constraint that these tools will introduce new functionalities at different level of the architecture, without introduce overlapping, exploiting their differences in goals, constraints, performance metrics and so on. Figure 1 shows how the different chosen prototypes will cover the Hybrid HPC-Cloud architecture defined in D4.FL3: the figure shows that there are not level of the architecture that have been not covered by at least a prototype, but also that the coverage is not homogeneous, although we tried to select prototypes to foster such characteristic; we can note that the most part of the prototypes cover the higher two levels of the architecture, while the orchestration and runtime management layers are less covered, in particular since the beginning: such functionalities are added through partners' frameworks such as the Edge/Cloud infrastructures described by the Politecnico di Torino, the University of Padova or the University of Pisa.

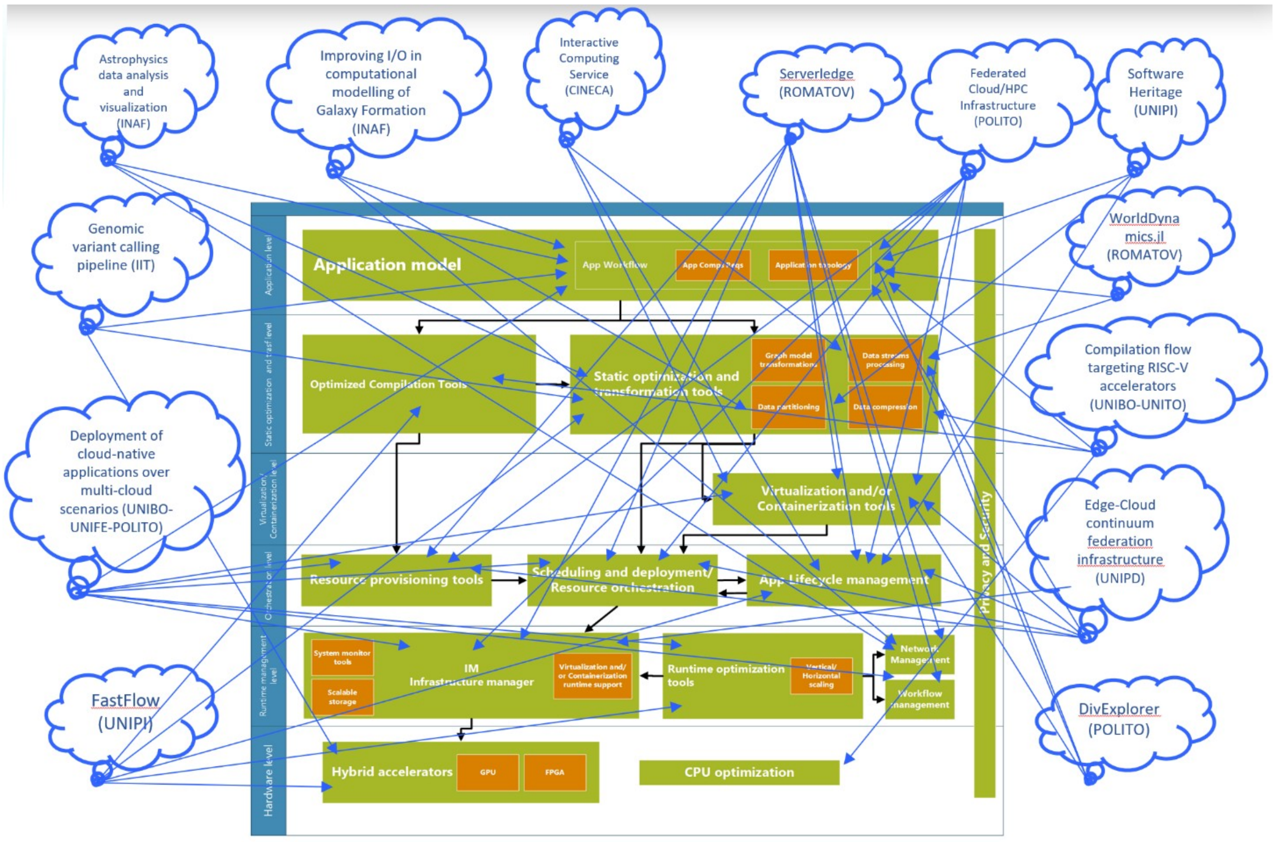
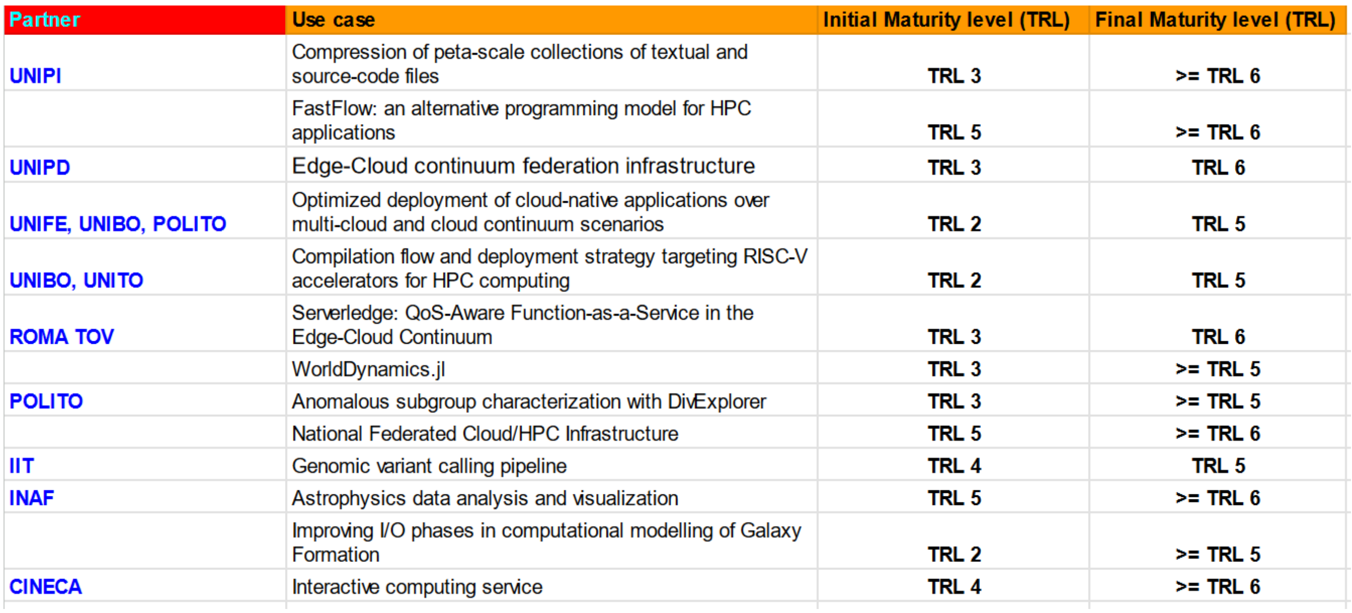

Astrophysics data analysis and visualization (INAF)
Improving I/O in computational modelling of Galaxy Formation (INAF)
Interactive Computing Service (CINECA)
Serverledge (ROMATOV)
Federated Cloud/HPC Infrastructure (POLITO)
Software Heritage (UNIPI)
Genomic variant calling pipeline (IIT)
WorldDynamics.jl (ROMATOV)
Compilation flow targeting RISC-V accelerators (UNIBO-UNITO)
Deployment of cloud-native applications over multi-cloud scenarios (UNIBO-UNIFE-POLITO)
Edge-Cloud continuum federation infrastructure (UNIPD)
FastFlow (UNIPI)
DivExplorer (POLITO)
Application level
Static optimization and trasf level
Virtualization/ Containerization level
Orchestration level
Runtime management level
Hardware level
Application model
App Workflow
App Comp Reqs
Application topology
Optimized Compilation Tools
Static optimization and transformation tools
Graph model transformations
Data streams processing
Data partitioning
Data compression
Virtualization and/or Containerization tools
Resource provisioning tools
Scheduling and deployment/ Resource orchestration
App Lifecycle management
System monitor tools
Scalable storage
IM Infrastructure manager
Virtualization and/ or Containerization runtime support
Runtime optimization tools
Vertical/ Horizontal scaling
Network Management
Workflow management
Hybrid accelerators
GPU
FPGA
CPU optimization
Privacy and Security

Figure 1: Hybrid HPC – Cloud architecture layers and prototypes' placement

The last important selection criterion we decide to take in consideration for this document is the maturity level of the actual existed prototypes. To give an objective, immediate and measurable value to this criterion, we employ the well-known **Technology Readiness Level** standard concept. **Technology Readiness Level (TRL)** is a concept used to assess the maturity and readiness of a particular technology or software product. Originally developed by NASA, TRL is now widely adopted across various industries to evaluate the advancement and potential of technologies before they are fully implemented or deployed. In the context of software, TRL provides a framework for measuring the readiness of a software solution based on its development stage, performance, and associated risks. It helps stakeholders, such as developers, investors, and decision-makers, to assess the feasibility, reliability, and potential impact of adopting a specific software technology. The TRL scale consists of several levels, typically ranging from TRL 1 to TRL 9, each representing a different stage of technology maturity. The idea is to increase the actual maturity level to at least a TRL of 5 (*Technology validated in relevant environment*) or 6 for the prototypes declaring the most maturity levels (*Technology demonstrated in relevant environment*), until the end of the project, choosing first prototypes that allow the most increase in TRL, declared by the partners in the description of each prototype in section 2. We do not give a superior limit to the final TRL, but a minimum final value of 5 for the TRL is needed and mandatory, to lead to a deployable prototype, allowing to test its main functionalities and to stress the integration of the tools in suitable environments such as the Edge/Cloud infrastructures referred before in this introduction. Such tests, described briefly at the end of every prototype's description section, try to validate a set of KPIs (decided by each partner) to stress the and involving, in some cases, very large datasets.



Partner	Use case	Initial Maturity level (TRL)	Final Maturity level (TRL)
UNIPI	Compression of peta-scale collections of textual and source-code files	TRL 3	>= TRL 6
	FastFlow: an alternative programming model for HPC applications	TRL 5	>= TRL 6
UNIPD	Edge-Cloud continuum federation infrastructure	TRL 3	TRL 6
UNIFE, UNIBO, POLITO	Optimized deployment of cloud-native applications over multi-cloud and cloud continuum scenarios	TRL 2	TRL 5
UNIBO, UNITO	Compilation flow and deployment strategy targeting RISC-V accelerators for HPC computing	TRL 2	TRL 5
ROMA TOV	Serverledge: QoS-Aware Function-as-a-Service in the Edge-Cloud Continuum	TRL 3	TRL 6
	WorldDynamics.jl	TRL 3	>= TRL 5
POLITO	Anomalous subgroup characterization with DivExplorer	TRL 3	>= TRL 5
	National Federated Cloud/HPC Infrastructure	TRL 5	>= TRL 6
IIT	Genomic variant calling pipeline	TRL 4	TRL 5
INAF	Astrophysics data analysis and visualization	TRL 5	>= TRL 6
	Improving I/O phases in computational modelling of Galaxy Formation	TRL 2	>= TRL 5
CINECA	Interactive computing service	TRL 4	>= TRL 6

Figure 37: Technology readiness levels of prototypes

In figure 37 are shown, for each of the thirteen prototypes selected for this document, (1) the main partner that had developed it until the beginning of the project, and has/have the main responsibility for their development as described in the section 2, for the rest of this project, (2) its denomination inside the project, (3) its actual TRL and (4) the declared TRL at the end of the project; we can note that the most number of prototypes start from a TRL of 2, 3 or 4 (with only 3 prototypes at TRL 5), allowing the possibility to increase each of them of at least two technology readiness levels, which is a sensible increase in its maturity.

2 Candidate prototypes

2.1 Compression of peta-scale collections of textual and source-code files

2.1.1 Introduction

Most organizations store and analyze larger and larger datasets that are often stored in public clouds such as Google Cloud, Amazon AWS, and Microsoft Azure. An important trend in cloud data warehousing is the separation of storage and compute, where the data is stored in distributed cloud objects, and where compute power can be spawned elastically on demand. The main idea of this emerging paradigm is to store data in so-called data lakes [88]: namely, storage systems with a blob API that hold data in generic and usually open formats, such as Apache Parquet and ORC. Blobs can be seen as any structured, semi-structured, and unstructured piece of data, such as source code files, row groups of databases, etc.. Compressing these large collections of files is very challenging and this problem was addressed in the past with various techniques, most of them spurring from a different but related problem, known as near-duplicate document detection. This arises mainly in the context of Web crawling, because duplicate and near-duplicate web pages induce significant drawbacks in the performance of Web search engines given their impact on index-space usage and on possibly returning repeated results. Thus, with the explosive increase in the size of the Web, search-engine designers started already in the '90s to investigate strategies for detecting these (near) duplicate pages. It was soon clear that a naive algorithm comparing all pairs to documents was prohibitively expensive, so researchers proposed algorithms for detecting near-duplicate documents with a reduced number of comparisons based on the concept of “fingerprinting”. After these results, the literature flourished on theoretically grounded approaches to fingerprinting methods and several experimental results were published to compare them. The A3lab of UNIPI has a long-time expertise in designing data compressors, with tens of papers concerning this topic and the related one of compressed indexes. In [87] we addressed the problem at hand via the so-called PPC paradigm: Permuting + Partition + Compress, whose main algorithmic idea was to first permute the files to bring close to each other the most “similar” ones; then partition them into blocks (of a proper size); and eventually compress each block with a suitable compressor (whose compression window is at least larger than the block size). At the best of our knowledge, no open-source software is available for compressing large collections of textual and source-code files.

In this project, we aim at designing and implementing a software library allowing to compress collections of several billions of texts and source code files (written in markup and programming languages, thus not just HTML) fully exploiting the computational power of the PPC-paradigm to achieve effective compression ratio and efficient (de)compression speed in two different scenarios: Backup and RandomAccess. The former concerns the storage scenario in which we support only a streaming access to the whole compressed collection; the latter concerns the case in which we want to support efficient access to individual files of the compressed collection. We plan to test our library on the Software Heritage archive [89] whose size is more than 1 PB of data, and it is continuously growing.

2.1.2 Related works

Open formats (like Parquet or ORC) internally use combinations of light-weight compressor schemes and general-purpose compression schemes, like LZ4, Brotli, Snappy or Zstd. This combination is neither efficient in terms of scan performance nor compact, and this is why new open-source file formats are emerging. Essentially, in all the mentioned solutions files (blobs) are always compressed individually trying to leverage the usually limited repetitiveness present inside each one of them. But, it is well known that these rapidly growing datasets are highly repetitive among blobs. In fact, blobs coming from the same context (i.e., same data lake) are often very similar, so it is essential to use proper compression techniques to leverage this characteristic and achieve high compression/decompression speeds over large data lakes.
This problem is like the near-duplicate document detection problem, as we commented in section 2.1.1. A naive algorithm comparing all pairs to documents is prohibitively expensive, so Manber [79] and Heintze [80] were among the first to propose algorithms for detecting near-duplicate documents with a reduced number of comparisons based on the concept of "fingerprinting". After these results, the literature flourished on theoretically grounded approaches to fingerprinting methods, with two pioneering and ground-breaking results by Broder [81] and Charikar [82]. Broder proposed to estimate the similarity of two documents by properly comparing a subset of the fingerprints computed from every sub-sequence of adjacent tokens, called "shingles", within the input documents. The obtained subset was called MinHash of the document. Charikhar proposed another approach, nowadays called SimHash, that estimates the similarity of two documents by randomly projecting each token of a document into a binary array, and then adding the projections of all its tokens. For both algorithms, there can be false positives (non-near-duplicate document pairs returned as near-duplicates) as well as false negatives (near-duplicate document pairs not returned as near-duplicates). Several papers (see e.g., [80], [83], [84], [85]) compared these two fingerprinting methods experimentally over collections of several billions of Web pages and declared SimHash as a robust practical approach. The literature offers other approaches to compute the set of fingerprints, the most notable one is Winnowing [86], which were proved to achieve better mathematical guarantees than SimHash and other interesting properties.
We contributed to this literature, by adapting those approaches to the case of HTML document collections [87] and proposing the so-called PPC paradigm: Permuting + Partition + Compress, whose main algorithmic ideas were described above. However, at the best of our knowledge, no open-source software is available for compressing large collections of textual and source-code files.

2.1.3 Actual prototype description and maturity level

Our preliminary prototype is based on the permute-partition-compress (PPC) framework, it hinges on a single-threaded implementation in Python, and it is able to manage GBs of data, but it is not able to scale to TBs/PTs. Its algorithmic structure consists of three main modules:

- [PERMUTE] create an ordering between the files/blobs;

- [PARTITION] partition the files into proper size blocks (typically of few MBs);
- [COMPRESS] compress all those blocks via a proper commodity general-purpose compressor (à la gzip, zstd, brotli, …). In order to support the random access to individual files, save for each of them the pointer to the compressed block where it is stored.

As far as the PERMUTE step is concerned, we have investigated two main approaches that we can classify as context-based or content-based, which appear to be very promising in the use case scenario we are considering.

The context-based approach deploys information coming from the filename, the file type, and the (parent) directories that include the file to be compressed; here, we postulate that these metainformation provide a proxy of the file content and thus can be exploited to cluster together similar files.

The content-based approach deploys information present inside the file to be compressed and thus requires sophisticated techniques that allow to manage TBs of data in succinct space and efficient time. Here, as explained in the Deliverable1 document we foresee the use of advanced shingling techniques and proper locality-sensitive hashes, together with efficient algorithms for the clustering of high-dimensional vectors (that spur from those hashes).

We are experimenting several instantiations of the PPC paradigm: from the simplest one, in which the permutation is the arbitrary one and the compressor is gzip (the one currently adopted in the Software Heritage archive); to more sophisticated approaches in which the permutation is based on the clustering of SimHash or MinHash fingerprints thanks to algorithms which exploit geometric or graph considerations; and, finally, we are investigating also the use of compressed indexes (à la FM-index or CSA) in order to achieve entropy-bounds in space occupancy and still preserving the ability to decompress only the requested file, and not much more.

Another dimension of our tests has been to evaluate our proposed compression methods in two different scenarios: Backup and RandomAccess. The former concerns the storage scenario in which we support only a streaming access to the whole compressed collection; the latter concerns the case in which we want to support efficient access to individual files of the compressed collection.

We have performed a very preliminary experimental analysis over a collection of millions of files derived from a snapshot of the most popular repositories on GitHub for a total of 25 GBs and have achieved significant compression ratios (up to 6% over C files and up to 15% over Python files) and (de)compression speeds (up to hundreds of MBs/sec), which compares favorably with respect to the current ratio of about 50% achievable on single-file compression.

The figure 2 reports the main result concerning the BackUp scenario over the collection of Python files, comparing [left] the compression speed (MB/sec) versus the compression ratio (%), and [right] the de-compression speed (MB/sec) versus the compression ratio (%).

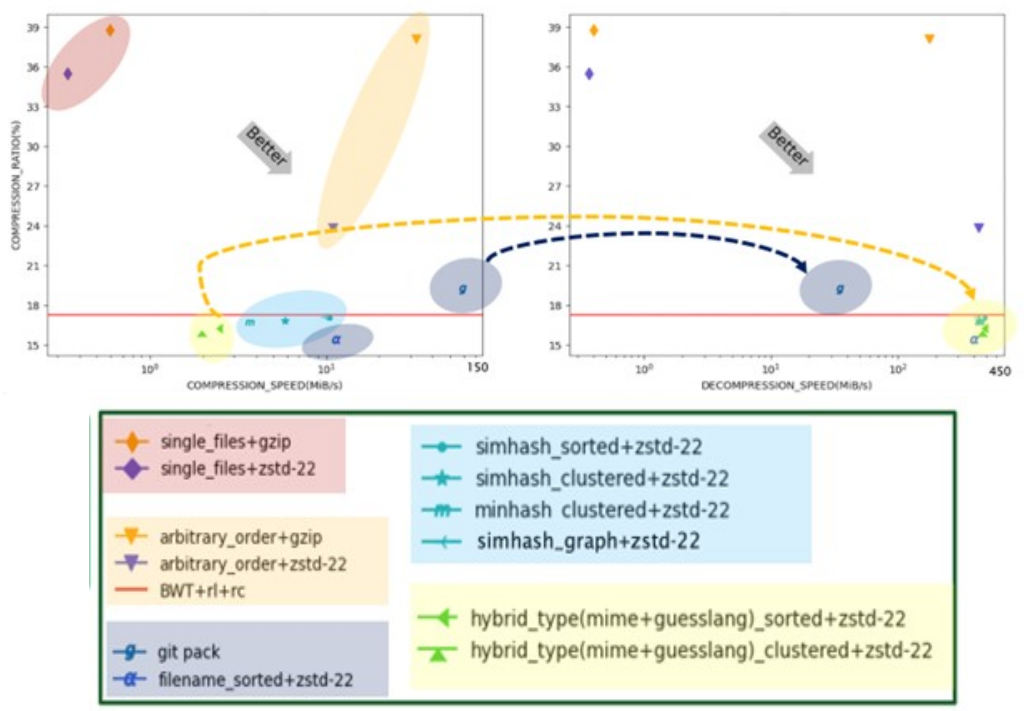


Figure 2: (De)Ccompression speed vs ratio

They include compression algorithms implementing context-based and content-based approaches, using as commodity compressors either gzip (with option -9) or zstd (with option -22), and working on:

- [red] individual files
- [orange] a sequence of files serialized according to a random order
- [blue] a sequence of files serialized according to the filename order
- [cyan] a sequence of files serialized according to an order established according to some fingerprinting methods combined with some clustering/sorting algorithms
- [yellow] a sequence of files serialized according to an "hybrid" order that exploits content-information derived from the MIME type and coding language of the file

Figure 2 clearly shows that the most powerful algorithms are able to achieve a compression ratio performance close to 15%, but with a compression speed which is very low, and thus possibly unacceptable for large file collections as the ones we wish to manage eventually. It is also evident that the very simple ordering based on filenames is among the most effective. The part on the right of Figure 2 shows that the decompression speed is very appealing and close to 500 MB/sec for a single-thread, and with the filename-sorted approach among the fastest ones. This performance is already interesting for a production scenario.
If we consider the results about C files, the compression ratio gets even more astonishing, because it reaches 6%. This compares very favorably against the gzip-approach on individual files which gets close to 30%.
As far as the RandomAccess scenario is concerned, the following picture shows the results for the case of files blocked in 4MBs or 16MBs, and with the balloon on the right picturing the differences with respect to the approach which compresses the whole serialized list of ordered files, hence without blocking.

It is evident that the loss induced by compressing the sequence of files in blocks is not large in terms of compression ratio, and this occurs even for the simplest serialization approach based on filename sorting. In the light of these experiments, we can classify the TRL 6 of our prototype as 3 (three): Experimental proof of concept.

2.1.4 Prototype evolution and implementation

The entities involved in the evolution of the prototype and its implementation will be mainly UNIPI and UNICS, with some contribution by IIT with respect to the compression of -omics data, as detailed below. Moreover, we expect synergies with the Software Heritage consortium (at INRIA), ENEA (which will host in the next months a mirror of the Software Heritage archive), and the Department of Computer Science at the University of Bologna.
The mirror is named "experimental" because its architecture and components differ from those adopted on the central archive at INRIA. Specifically, the mirror storage employs the distributed file system Seaweedfs, which is designed for efficiently storing a huge number of small files, such as source codes generally are. Furthermore, by "experimental" it is also intended that the mirror is not a plain replica of the main archive at INRIA, but relies on a partially different technological implementation, better suited to the particular type of 'data' handled and to its processing for research purposes and well integrated into ENEA's HPC facility.
We plan to scale the above preliminary experimental scenario to 200 GB first, and later to TBs of data. This data will be fetched from the largest available archive of open-source code — i.e., Software Heritage — thus contributing to its long-term code preservation with reduced HW resources. The Software Heritage [89] initiative aims at collecting the complete history/heritage of human coding publicly available, replicating it massively to ensure its preservation, and sharing it with everyone who needs it, from science to industry. The initiative was launched in 2015 by INRIA – the French national research institute for digital science and technology – in agreement with UNESCO and assembling a group of prestigious supporters and committed sponsors including, among the others, Microsoft, Intel, Google, VMWare, GitHub, Qwant, Nokia Bell Labs, Société Générale, the Alfred P. Sloan Foundation, and the Universities of Bologna and Pisa. Our university is part of this initiative both formally and operationally because in the last few months we have started collaborating on the design of the storage and indexing infrastructure of the Software Heritage dataset. We aim at addressing this large-scale problem as a UseCase of this PNRR project because of its massive size, its novelty and impact, as well as prestigious collaborations that we will be able to set out of it.
Figure 3 provides a snapshot of the evolution of the Software Heritage archive since its creation in 2015.

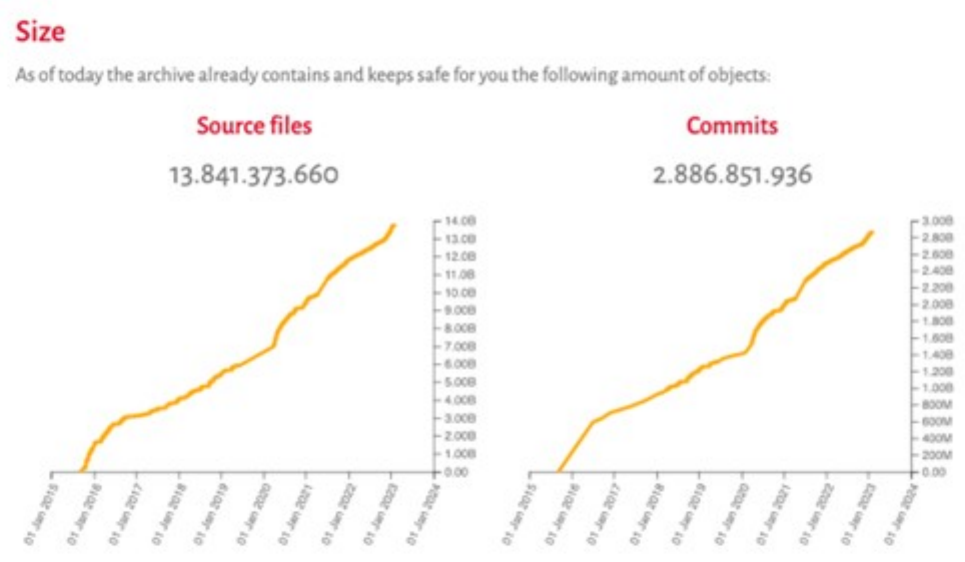


Figure 3: size of the Software Heritage archive

In technical terms, the Software Heritage dataset is organized as follows. Source code artifacts (e.g. software projects, releases, commits, directories, etc.) ingested by the crawlers are stored and organized as a direct acyclic graph whose leaves are the so-called "blobs", which represent the raw content of (source code) files. Presently, blobs contribute to about 99% of the space of the main copy of the Software Heritage archive, which is reported to contain over 800 TB of data [89]. Since the archive is steadily growing, the consequent impact on the scalability and storage cost of the archive and its mirrors is becoming a serious concern, not only in economic terms, but also in terms of energy demands and environmental impact of operating storage devices and replacing them when worn-out.

To mitigate this issue, blobs are currently compressed individually using the classic gzip tool and thus achieving a compression ratio which is around 50% (hence, half of the archive). However, this approach fails to exploit the vast compression opportunities offered by new compressors (such as Brotli, Zstd, etc.) and by the redundancies present in the collection of files belonging to this archive. There are indeed intra-repository redundancies (e.g. versions of the same source code file) and inter-repository redundancies (e.g. different repositories implementing similar software components in the same programming languages). As reported above, the compression of individual files using gzip is roughly a factor of 2x; however, if we use zstd (one of the best performing compressors to date) with the most time-consuming parameter settings the compression ratio gets to around 25% with a factor 2x improvement. However, both these approaches do not exploit the intra-repository redundancies and inter-repository redundancies known to exist in the Software Heritage archives. The experiments conducted in the previous section over a small collection of 25GB of files coming from GitHub, have shown that our preliminary prototype can achieve significantly improved performance in several respects: compression ratio, compression speed and decompression speed.

Therefore, the use case we plan to address consists of deploying our newly designed compression tools, possibly extending and improving them (in efficiency and efficacy), to compress a large part of or the entirety of the Software Heritage archive, taking as much as possible advantage from both the context-based approach and the content-based approach. Our aim is to establish which one provides the best compression ratio in this scenario at reduced computational resources.

The current status of our prototype is in terms of a single-threaded implementation. New (single-threaded) algorithms are in preparation to implement versions of the context-based and the content-based compressors and will be designed and implemented in the next few months for achieving an even better compression ratio. This code will be distributed via GitHub or other code-sharing platforms, and it will constitute, as far as we know, the first open-source library for compressing peta-scale collections of textual and source-code files.

A parallel and distributed implementation is to be developed to achieve better time efficiency, given the significant size of the Software Heritage archive. There are two key issues to investigate. First, we need to understand whether the context-based features are enough to derive effective compression ratios because they are easily computable and exploitable (being based on the sorting of short strings), and therefore they can easily scale to GBs/TBs archives by deploying parallel/distributed sorters (such sorter which are available in most BigData platforms). Second, we need to investigate how to efficiently compute the content-based features, which require a high throughput for the shingles and highly-scalable clustering and sorting algorithms for grouping similar files.

We intend to extend our compression libraries to a parallel and distributed scenario by adopting models, frameworks, and tools for parallel and distributed batch processing across clusters. We therefore foresee synergies in the development of our compression pipeline with the following libraries: **ParSoDA** - Parallel Library for Big Data Analysis (UNICAL) (see D4.FL3, section 2.2.2) and the library **FastFlow/WindFlow** for high-level and efficient streaming (UNIPI-DI) (see D4.FL3, section 2.2.6), and with **Genomic variant calling pipeline** (IIT), described here in section 2.3. Furthermore, we are evaluating the feasibility and the hardware requirements (in terms of networking, storage, and computing) to transfer, store and compress a mirror of either a large part or the entirety of the Software Heritage archive.

ParSoDA is a parallel computing library that includes algorithms widely used to process and analyze on multiple computing nodes data gathered from different sources (e.g. web, social media) with the goal of extracting different kinds of information (e.g., user mobility, user sentiments, topic trends, frequency, etc.). ParSoDA defines a general structure for a data processing pipeline that is composed of different steps (data acquisition, data filtering, data mapping, data partitioning, data reduction, data analysis and data visualization) that can be combined together based on the application logic. For each of these steps, ParSoDA provides a predefined set of functions. For example, for the data filtering step, ParSoDA provides functions for filtering geotagged items based on their position, time of publication, and contained keywords. Programmers are free to extend this set of functions with their owns. The current version of ParSoDA contains a wide set of predefined parallel functions organized in seven packages, corresponding to the seven ParSoDA steps. Details on each function are available at [90]. Applications based on the ParSoDA library can be run on both Hadoop and Spark clusters. This allows ParSoDA to reduce the execution time by parallelizing code execution and exploiting the computational and storage resources of clusters.

In the context of this use case, ParSoDA can be used to define a data compression pipeline composed of three main phases: 1) parallel sorting of files based on their filenames, or other content- or context-based features; 2) serialization and grouping of files in blocks of predefined size; 3) parallel compression of those blocks of files by commodity or ad-hoc compressors. The first phase will be implemented by defining an ad-hoc function that exploits parallel and distributed (string) sorting algorithms. For the second phase, we will investigate how to exploit data locality to reduce data migration costs. Finally, a data parallel function will be defined to perform the data compression phase. An experimental evaluation will be performed to select the most appropriate runtime (Hadoop/Spark) and file system (e.g., HDFS) to be used.

FastFlow is a C++ header-only parallel library intending to provide application designers with essential features for parallel programming via suitable abstractions (e.g., Pipeline, ordered Task-Farm, Divide&Conquer, Parallel-For-Reduce, Macro Data-Flow, Map-Reduce) and a carefully designed runtime system. At the lower software layer of the library, there are the so-called Building Blocks (BBs), i.e., recurrent data-flow compositions of concurrent activities working in a streaming fashion, which are used as the primary abstraction layer for building high-level parallel patterns and complex data-streaming network topologies of sequential operators. Following the principles of the structured parallel programming methodology, a parallel application (or one of its components) is conceived by selecting and assembling a small set of well-defined BBs modeling data and control flows. BBs can be combined and nested in different ways forming either acyclic or cyclic concurrency graphs, where nodes are FastFlow’s concurrent entities and edges are communication channels. All high-level parallel patterns offered by the upper layers of the FastFlow library have been implemented using the BB components. Furthermore, BBs have been used to implement special-purpose parallel libraries providing high-level APIs for specific application domains. A remarkable example in this sense is the WindFlow library for Data Stream Processing, which has been built upon FastFlow’s BBs with extensive support for different streaming semantics (e.g., timestamped ordered execution, watermark propagation, and micro-batching) and the seamless use of HW accelerators (GPUs and FPGAs). In terms of support for different architectures and platforms, FastFlow provides the application programmers with a unified interface that enables them to generate executable programs that can run on both shared- and distributed-memory platforms. The distributed runtime has been implemented by leveraging BBs and extending them to preserve the original data-flow streaming programming model. Inter-process communications are implemented both in TCP/IP and MPI.

In this use case, the FastFlow library can be used to implement the entire data compression workflow expressing stream parallelism between the different phases, thus enabling their overlap. Additionally, leveraging the FastFlow parallel BBs can accelerate the single workflow phase to improve vertical scalability by enforcing, if needed and suitable, the WindFlow library to accelerate intra-node parallelization phases requiring complex streaming semantics and/or the use of HW accelerators. However, careful implementation and an in-depth experimental analysis will be needed to identify a suitable trade-off between the different kinds of parallelism used to maximize the utilization of computing resources.

For these purposes, ENEA makes its storage resources and CRESCO6 supercomputing infrastructure available to the project. CRESCO6 - an HPC cluster of 434 nodes with 2 Intel 24-core processors and a GPFS parallel filesystem - is integrated into the ENEA grid environment and is located in Portici. Furthermore, thanks to a dedicated 10Gb link between ENEA Bologna and ENEA Portici, a large part of the Software Heritage blob archive could be also replicated on the high throughput storage of CRESCO6 to sustain the increased processing power of the parallel and distributed implementation.

2.1.5 Final validation tests

The *KPI* used to evaluate the success of our tests will be the classic ones in the realm of data compression tools: compression ratio, compression speed (MB/sec) and decompression speed (MB/sec), as adopted in the previous section.
A first milestone of this project will be the creation of a few datasets on which to experiment our compression tools. The datasets size should be of up to a few hundred GBs, so that they will provide a succinct representation of several facets of the whole archive. In particular, we will aim at downloading snapshots of that size for the various coding languages (C, Python, Javascript,...) and also a random snapshot mixing several languages, in order to have a better “picture” of the performance of our tools over the overall archive through the use of a sample of reduced size. This should allow us to extrapolate performance figures for the whole archive and for the more homogeneous parts consisting of blobs containing a single coding language.
As of September 2022, the main copy of the Software Heritage archive contains about 12 billion blobs, with a median size of 3 KB, for a total compressed size of about 800 TB [89]. Since the compression technique that is currently adopted in Software Heritage is reported to obtain an average compression factor of 2x (by applying gzip on individual files), we can expect that the total uncompressed size of the blobs is about 1.6 petabytes. If the preliminary results obtained on the snapshot of 25GBs (shown above) would be confirmed at the total archive scale, then the trivial use of the better zstd (in place of gzip) would guarantee an occupancy of 400TBs. But in this use-case we aim for more, and thus hope to get down to a few TBs of space occupancy for the whole archive, or to achieve few TBs (or less) of space occupancy for “vertical” archives in which we have only Python, Java, Javascript, C or C++ source codes.
In addition to the case of the Software-Heritage archive, we will also investigate the application of this software library to the case of genomic datasets, such as VCF files, which are growing in size because of the continuously improving performance of modern DNA sequencing machines. In this context some codes are already available – say genozip, vcfshark, etc. – we aim at understanding how they can be combined with our approach, as we did for gzip and zstd, and working on collections of files rather than single ones. In this context we expect that content-based

approaches will perform better than context-based ones (i.e. filename sorting) because these latter are mainly agnostic to the file content.

2.2 Astrophysics data analysis and visualization

2.2.1 Introduction

Modern astronomy and astrophysics produce massively large data volumes (in the order of petabytes) coming from observations or simulation codes executed on high performance supercomputers. Such data volumes pose significant challenges for storage, access and data analysis, leading to the development of a fourth data intensive science paradigm [1] (also known as eScience). A critical aspect in understanding, interpreting, and verifying the outcome of automated analysis and data mining processes is the visualization of the scientific results. Data visualization is a fundamental, enabling technology for knowledge discovery, and an important research field that covers a number of different topics such as: optical and radio imaging, simulation results, multi-dimensional exploration of catalogs and public outreach visuals.
Visual exploration of big datasets poses some critical challenges that must drive the development of a new generation of graphical software tools [2], specifically:

- *Interactivity*: The majority of existing astronomical analysis and processing solutions lack the ability to deal with datasets exceeding the local machine's memory capacity while visual exploration and discovery in complex, multi-dimensional datasets is more effective through real-time interaction although sizes may not fit the available memory. For complex visualizations the relevant computations should be performed close to the data to avoid time consuming streaming of large data volumes. This can be achieved via flexible distributed architectures striking a balance between local interactive exploration tools and remote services hiding data complexity.
- *Integration*: Most of the data analysis systems are implemented as a set of separate independent tasks that can interact and exchange information via stored files only. This will be a significant factor which delays or even prohibits day-to-day data analysis tasks over big data sizes. Visualization tools should be ideally fully integrated within the scientists' toolkit for seamless usage, abstracting from technical details freeing scientists to concentrate in doing science. Tools should be coupled with advanced high performance computing (HPC) resources to deal with requests to archives through scientifically meaningful lightweight versions of the datasets obtained by analysis/processing operators since full data sizes may not fit the available memory to allow real-time interaction.
- *Navigation*: Some of the current data processing techniques depend on parameters tuning, which may not be easy to achieve with large data sizes due to processing power limitations. Adopted solutions should allow intuitive and sophisticated navigation among datasets by exploiting ubiquitous environments, such as tablets or motion controllers, offering new Human–Computer Interaction paradigms to better tune the processing parameters. Local exploration tools should enable interactive visualization optimized for ubiquitous computing environments, intuitively controlling the resulting visualization.
- *Collaboration*: It will no longer be an easy job to develop a simple script or program to deal with such data. These tasks usually require a deep

knowledge of technicalities and programming experience which are typical of computer scientists rather than of astronomers. Tools should be built into the processing pipelines in order to facilitate visualization, processing and analysis of big data in a collaborative manner. Tools should be combinable within e.g. science gateway technologies to allow collaborative activity between users and provide customization and scalability of data analysis/processing workflows, hiding underlying technicalities.

Therefore, over the years, the astrophysics domain has developed a set of ad-hoc tools and software modules to tackle these challenges. With the emergence of high-performance visualization and Visual Analytics (VA) as enabling technologies, some of these components become candidates to be replaced by either faster, more accurate, or more efficient data-driven technologies modeling pre-processing, run-time, and post-processing stages by exploiting the latest technological opportunities.

2.2.2 Related works

Visualization plays a fundamental role in almost every scientific discipline facilitating qualitative and quantitative data analysis, for new knowledge generation and effective communication of end results. Suitable tools and approaches can boost scientific productivity significantly, e.g., by revealing hidden trends or intrinsic patterns in the data, leading to fresh insights and eventually, new scientific discoveries.

The big data revolution is providing enormously large, incredibly rich, and highly complex data volumes that impose extremely challenging demands on traditional visualization approaches (see e.g. [17, 18]). The demands to address are efficiency, i.e. the ability to handle rapidly the underlying data complexity, and intuition, i.e. the ability to reach suitable interpretation by domain experts.

Innovative visualization tools and solutions must be able to (i) handle complex and heterogeneous datasets, (ii) support multiple visualization strategies (e.g., 2D and 3D renderings, projection techniques for higher dimensionality data) and (iii) enable an intuitive and user-friendly data exploration. Moreover, the ever-growing size of the datasets underlines the need for moving from the traditional standalone model to novel distributed approaches, relying on cloud-based infrastructures able to meet the increasing demand for resources.

ParaView [19] is a large scale parallel visualization software, designed for effective exploitation of high performance infrastructures. A web enabled version, ParaViewWeb, can act as a Web Application by allowing users to remotely connect via web browsers to a ParaView server. The Cactus computational framework [20] can support a web browser interface for in situ visualization and steering tasks. The user can instrument existing high performance applications with the Cactus API, perform steering tasks and view visualization outputs through a web browser. WebVis [21] is a multi-user, client-server, visualization framework with a web-based client offering services in the cloud and is accessible via netbooks, smartphones, and other web-and JavaScript-enabled mobile devices.

2.2.3 Actual prototype description and maturity level

INAF has been developing and maintaining the Visualization Interface for the Virtual Observatory (VisIVO) [4,23] at the TRL5 level and extended it with the ViaLactea Visual Analytic [5] module (VLVA).

VisIVO is developed adopting the Virtual Observatory standards and its main objective is to perform 3D and multi-dimensional data analysis and knowledge discovery of a-priori unknown relationships between multi-variate and complex astrophysical datasets.

VisIVO is deployed in a variety of flavours as follows:

- VisIVO Server [5] - a platform for high performance visualization,
- VisIVO Library [5]- for running complex workflows on DCI, clouds and HPC infrastructures to efficiently produce complex views of the dataset and full movies directly with the user-code internal data representation (i.e. without the need to create intermediate files).
- VisIVO ViaLactea Visual Analytics (VLVA) [6,7] - which allows to exploit a combination of all new-generation surveys of the Galactic Plane to analyze star forming regions of the Milky Way.

a. Prototype modelization, structure and functional description

To render the visualization, we typically require three steps: data importing, filtering and viewing. The importing process converts the supplied datasets (originally in different formats) into an internal binary format. A VisIVO Binary Table (VBT) is a highly-efficient data representation used by VisIVO Server internally. A VBT is realized through a header file (extension .bin.head) containing all necessary metadata, and a raw data file (extension .bin) storing actual data values. For example, the header may contain information regarding the overall number of fields and number of points for each field (for point datasets) or the number of cells and relevant mesh sizes (for volume datasets). The raw data file is typically a sequence of values, e.g. all X followed by all Y values. The filtering process allows to perform several operations on the data, this may include randomization or decimation to reduce the final resolution, mathematical or statistical operators or commonly adopted cosmological post-processing such as the three commonly used mass assignment functions, i.e., the nearest grid point (NGP), the cloud-in-cell (CIC), and the triangular-shaped cloud (TSC) methods. Finally the visualization process creates multi-dimensional views from the data that must fit the available RAM. The kinds of visualization include data points, volumes and vectors and are based on the Visualization TooKit (VTK) [8].
Figure 4 depicts the typical visualization pipeline of VisIVO Server consisting of the application of the three main modules: VisIVO Importer, one or more VisIVO Filter(s) and one or more VisIVO Viewer(s).

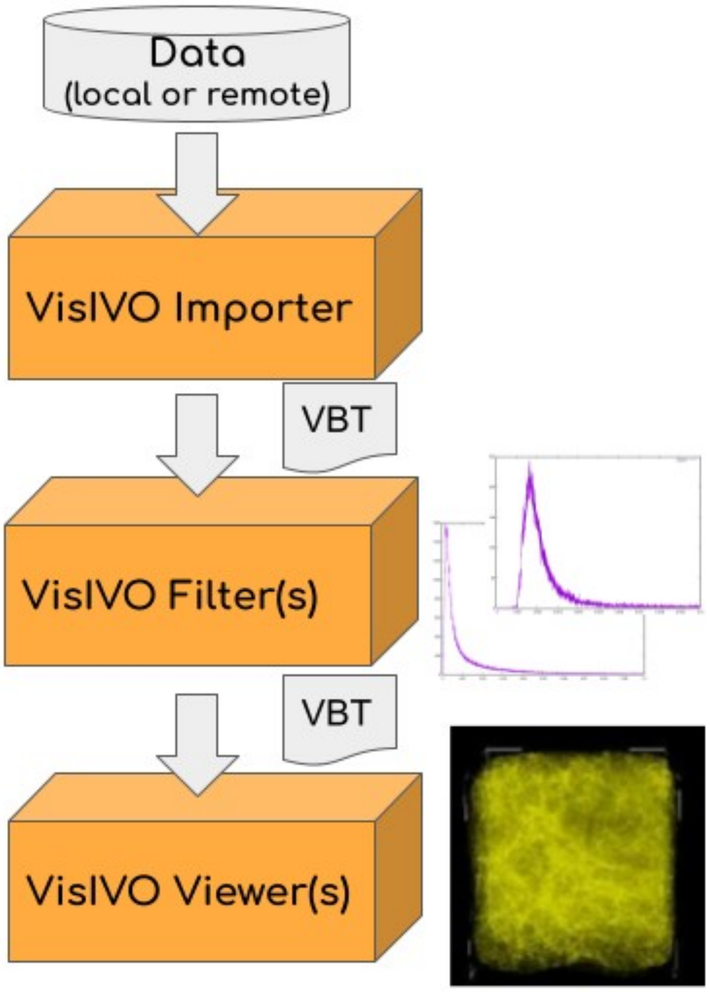


Figure 4:visualization pipeline of VisIVO server

b. Actual implementation

In the presented use case, we deal with multi-dimensional simulated and observational data. Simulated data are produced by N-body/hydrodynamical cosmological simulations. We will primarily focus on GADGET [2] (GAlaxies with Dark matter and Gas intEracT) simulated data. The primary result of a simulation with GADGET are snapshots, which are simply dumps of the state of the system at certain times. GADGET supports parallel output by distributing a snapshot into several files, each written by a group of processors. This procedure allows an easier handling of very large simulations; instead of having to deal with one file of larger size, it is much easier to have several files with a smaller size. Each particle dump consists of a multiple number of files. Each of the individual files of a given set of snapshot files contains a variable number of particles. However, the files all have the same basic format, and all of them are in binary. A binary representation of the particle data is our preferred choice, because it allows much faster I/O than ASCII files. In addition, the resulting files are much smaller, and the data is stored loss-less.
Observational data are mainly related to the major new-generation surveys of the Galactic Plane from the infrared to the radio band, both in thermal continuum and in atomic and molecular lines, from Europe-

funded space missions and ground-based facilities. We will primarily focus on large scale data cubes in the radio band coming from precursors and/or pathfinders of the Square Kilometre Array [3], the largest and most accurate radio telescope arrays which are under construction in Australia and South Africa.
The size and complexity of this data require optimized codes and full integration within the astronomical pipelines and workflows exploiting HPC and exascale systems.
Within the FL3 activities, VisIVO will be extended and optimized for the real-time visualization of cosmological simulated data, giving the opportunity to compare with observational multiwavelength data, exploiting the available HPC platforms.
The importing modules are being parallelized for multi node/multi thread platforms using MPI. Specifically, the importing modules will use MPI-IO to parallelize multiple reads and writes on common files and a Consumer-Producer approach, useful for load balancing, depicted in Figure 5.

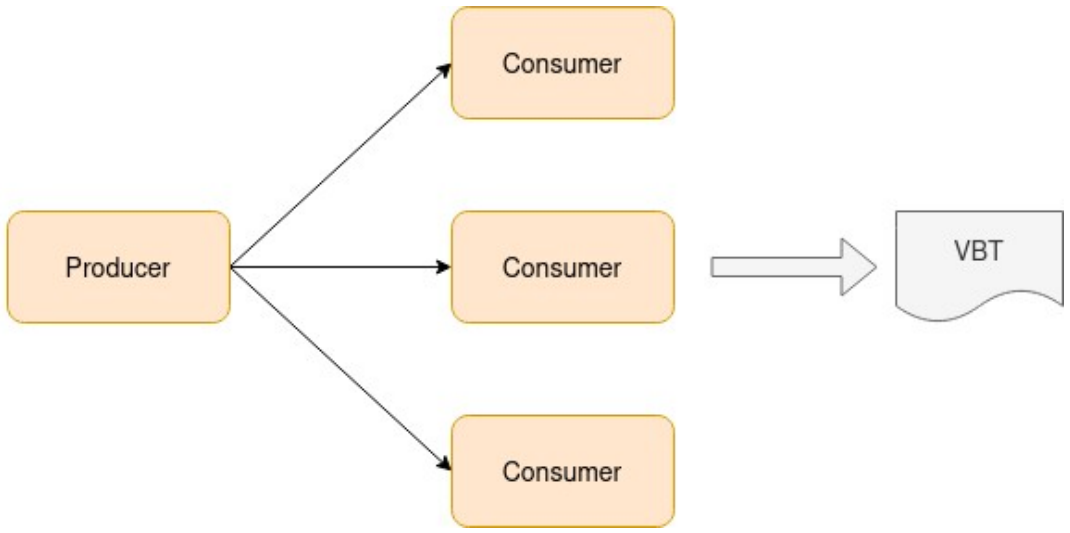


Figure 5:Producer - Consumer approach schema

The filtering modules will be extended to exploit multi GPU platforms investigating CUDA and OpenACC[10]. Depending on the underlying complexity of the filter modules, some of them will instead employ MPI (e.g. the filters to merge VBTs or add new tabular columns).
The viewer modules core technology, based on VTK, is already optimized for emerging processor architectures [11] and will be tailored to support the fine-grained concurrency for data analysis and visualization algorithms required to drive extreme scale computing by providing abstract models for data and execution that can be applied to a variety of algorithms across many different processor architectures. Finally a client-server based architecture (employing Paraview) will be exploited to avoid large scale data movements and to set up the render engine close to the data.

c. Validation tests and results

VisIVO has been already deployed using Science Gateways [22, 9] to access DCIs (including clusters, grids and clouds) using containerization and virtualization technologies, it has also been selected as one of the pilot applications deployed on EOSCpilot infrastructure demonstrating that the tools can be accessed using gateways and cloud platforms and it has been deployed on EOSC, efficiently exploiting Cloud infrastructures and interactive notebooks applications [9].

2.2.4 Prototype evolution and implementation

a. Prototype evolution direction

The final aim of the prototype evolution direction and the related implementation activities will be tailored to pursue the following objectives:

- **O1**: Enhance the portability of the VisIVO modular applications and their resource requirements. VisIVO modules, i.e. importer, filter(s) and viewer, are being parallelized to further exploit HPC and Exascale infrastructures in FL5. Thus improving its portability and potentials of integration with other astrophysical pipelines and computing resources will make VisIVO fully integrated within the scientists' toolkit for its seamless usage, abstracting from technical details freeing astronomers to concentrate in doing science.
- **O2**: Foster reproducibility and maintainability. A visualization aided data analysis usually requires several parameter settings for pre-processing and actual rendering of a complex multidimensional dataset. Furthermore, in this era of Open Science, offering novel mechanisms and techniques to make scientific discoveries reproducible and maintainable is a must, especially for enhancing scientific and technical collaboration.
- **O3**: Take advantage of a more flexible resource exploitation over heterogeneous HPC facilities (including also mixed HPC-Cloud resources). So far VisIVO has been onboarded to the European Open Science Cloud (EOSC) and has been improved toward meeting the Cloud requirements and offered services. With the increasing size and complexity of astrophysical datasets, there is a need for increasing the computing performances as well as the storage capacities for processing and analysis tasks while maintaining the Cloud software-as-a-service opportunities.
- **O4**: Minimize data-movement overheads and improve I/O performances. The importer modules of VisIVO rely on heavy I/O tasks for translating the astrophysical datasets in the internal VisIVO binary format that is also the one imported for the VisIVO filtering and visualization modules. Therefore, minimizing the computing costs of these I/O tasks could potentially improve the overall performances of the VisIVO pipelines.

The VisIVO tools and related software will be provided by INAF Astrophysical Observatory of Catania (OACT). For this work plan we expect collaboration with UNITO (for StreamFlow and Jupyter Workflow, see deliverable D4.FL3), with UNIPI (for CAPIO and Nethuns, see deliverable D4.FL3) and with CINECA (for the Interactive Computing Service, see D4.FL3, and for improving the parallel implementation of the VisIVO modules).

b. Prototype evolution structure and description

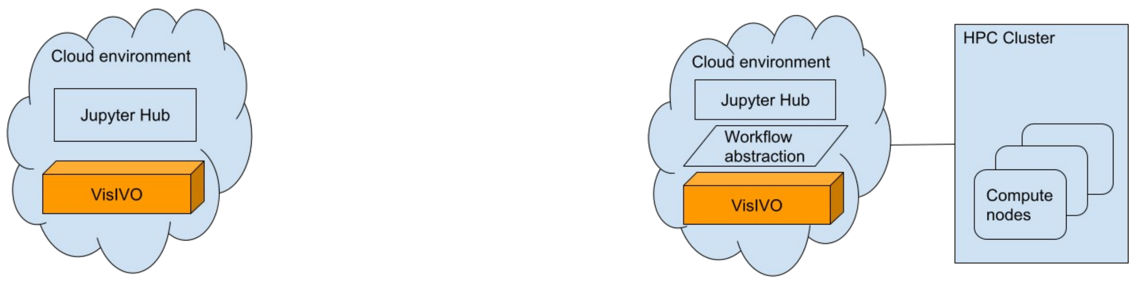


Figure 6: evolution of the Cloud deployment prototype

Additionally, we would like to improve I/O performances, since VisIVO workflows communicate through file read/write operations as depicted e.g. in the exemplificative workflow depicted in figure 7.

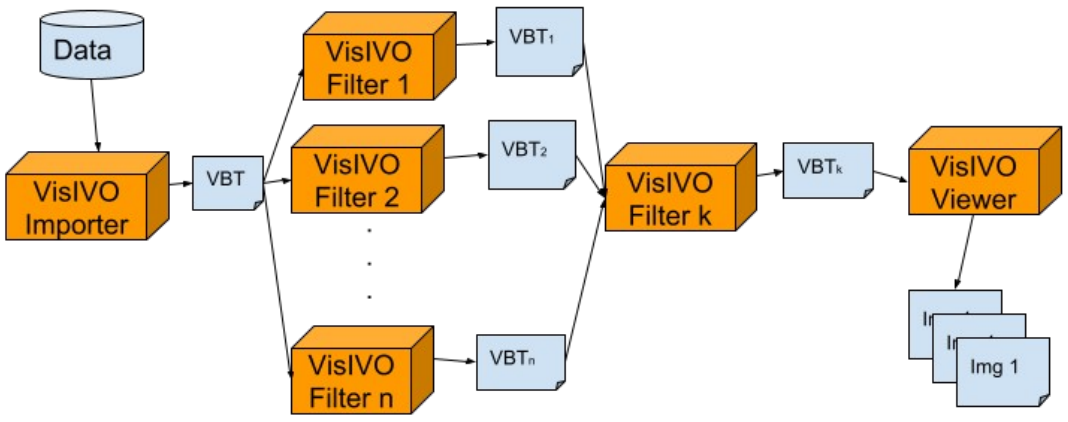


Figure 7: exemplificative workflow of VisIVO communications

c. Prototype implementation and involved tools

We would like to investigate workflow abstractions to allow a portable representation of the VisIVO modular applications and their resource requirements, fostering reproducibility and maintainability, to take advantage of heterogeneous HPC facilities (including also mixed HPC-Cloud resources) while minimizing data-movement overheads. In particular:

- ***StreamFlow (UNITO).*** We aim to: i) execute the different VisIVO modules in multi-container environments to eventually support the concurrent execution of multiple communicating tasks in a multi-agent ecosystem; and ii) to allow for hybrid workflow executions on top of multi-cloud or Hybrid Cloud HPC infrastructures. We expect that its hybrid workflow approach will enable the deployment of the different distributed VisIVO workflow steps onto different modules and to exploit the topology awareness emerging from the VisIVO workflow models allowing StreamFlow to implement locality-based scheduling strategies, automated data transfers, and fault tolerance. Moreover, we expect to increase the reproducibility and provenance of our VisIVO workflows also thanks to the exploitation of the Common Workflow Language [12] (CWL) and all its related platforms e.g. the Workflow Hub [13] and the RO-CRATE [14] and supports also other Workflow managers like Galaxy [15], Airflow [16] or others (see D4.FL3, section 2.1.6).
- ***Jupyter Workflow (UNITO).*** Additionally, we plan to investigate the Jupyter Workflow kernel to describe the VisIVO workflows and execute them in a distributed fashion on Hybrid Cloud HPC infrastructures aiming to improve the usability, readability and maintainability of VisIVO applications. Moreover, we expect this integration to improve the application scalability to better exploit the heterogeneity of the underlying computing resources (see D4.FL3, section 2.1.5).
- ***The Interactive Computing Service (CINECA).*** The service will be exploited to explore the functionalities offered in particular the ones related to the web interfaces enabling VisIVO pipelines. This work will include the implementation of Python wrappers to VisIVO Command Line Interfaces thus seamlessly integrating VisIVO with interactive notebooks and Python codes. Additionally, we will test and explore the VNC based features for enhancing the capabilities of VLVA and, finally, we will test the StreamFlow integration within the service when it will be available (see D4.FL3, section 2.1.4).

Additionally, we would like to investigate fast I/O techniques for optimizing the importing of large-scale datasets (currently employing MPI). Such as:

- ***CAPIO (UNIPI+UNITO).*** We would like to investigate the integration of VisIVO workflows with the CAPIO middleware to boost its I/O performances without modifying the original codebase and allow it to coordinate the I/O within the VisIVO modules and, eventually, inject streaming capabilities into its workflow (see D4.FL3, section 2.4.3).
- ***Nethuns (UNIPI).*** Additionally, we will investigate the feasibility to integrate the lightweight userspace library Nethuns that offers a straightforward programming model for network I/O and test the available I/O accelerations frameworks, nicknamed engines, as a backend (see D4.FL3, section 2.4.2)

2.2.5 Final validation tests

The final validation tests will be tailored to evaluate the degree of accomplishment of the development activities toward reaching the 4 objectives presented in Section 2.2.4 part *a* targeting the increasing of the TRL higher than TRL 6. The enhanced portability of Objective 1 and the increased flexibility of resource exploitation of Objective 3 will be validated by the successful implementation and execution of VisIVO workflows on a number of different computing infrastructures, including Cloud and HPC centres at INAF and also the others available by the FL3 consortium (e.g. HPC4AI from UNITO or infrastructures at CINECA) eventually exploiting ready to use interactive notebooks. To accomplish this latest activity, a python wrapper to VisIVO Server will be developed. The increased reproducibility and maintainability of the visualization pipelines offered by VisIVO, as mentioned in Objective 2, will be validated by a number of workflows developed and available on common repositories to demonstrate the results over challenging use cases such as large scale cosmological simulations or visualization of large scale SKA precursors datacubes. Finally I/O performances will be measured to validate the Objective 4 tested on datasets with increasing size and complexity.

To summarize, the Key Performance Indicators used to evaluate the success of our developments are presented in the following table.

KPI	Description	Success measure
VisIVO Workflows	The delivery of a repository including the most common workflows to exploit VisIVO in data intensive scenarios (e.g. cosmological simulations or visualisation of large scale SKA precursors datacubes) and perform the processing on different computing infrastructures (from HPC to Cloud).	At least 3
VisIVO wrapper	The development of a wrapper for VisIVO Server in Python to integrate VisIVO with interactive notebooks and Python codes.	3 (one for each VisIVO Server module: Importer, Filter, and Viewer)
VisIVO interactive notebooks	Integration of ready-to-use VisIVO notebooks templates to be exploited and eventually customised within Jupyter Workflow and/or the Interactive	At least 3

	Computing Service.	
I/O performances	The improvement of VisIVO I/O performances with CAPIO and, eventually, Nethuns tested on datasets with increasing size and complexity.	Measurements of I/O performance improvements over at least 3 different datasets with increasing size and complexity.

2.3 Genomic variant calling pipeline

2.3.1 Introduction

Omic data analysis is becoming more and more a routine activity in several hospitals and research labs. This data availability allows for an unprecedented amount of molecular detail which opens new avenues in terms of data analysis towards precision medicine [24]. When dealing with genomic data in particular, these data require a systematic and not trivial pre-processing step before any actionable knowledge can be derived. This step is generally referred to as variant calling and it is the one which bridges the raw data to information usable by the clinician to carry on the investigation and infer a possible disease or disease predisposition. Variant calling is indeed a critical step in genomic analysis that involves identifying genetic variations, such as single nucleotide polymorphisms (SNPs), insertions, and deletions, across the genome.
Genomic data coming from Next Generation Sequencing (NGS) devices [25] (or more recently even from Oxford Nanopore devices [26]) require indeed the definition of a dedicated multi-step pipeline for variant identification followed by varian annotation and prioritization. Towards this aim several intermediate tools must be used and employing workflow managers to orchestrate such pipelines is crucial to obtain an efficient and manageable execution. In this proposal we discuss the enhancement of an existing pipeline. In the following, in section 2.3.2, we revise some recent literature on the topic, in section 2.3.3 we discuss our actual prototype, in section 2.3.4 we discuss the planned improvements and in section 2.3.5 we devise a set of final tests for the enhanced platform validation.

2.3.2 Related works

Over the years, a wide range of variant calling algorithms have been developed, each with its own strengths and limitations. Traditional methods such as samtools [27] and GATK [28] rely on mapping reads to a reference genome and using statistical models to detect variants. These prove to be reliable and largely used. Together with these methods others try to improve the performance especially in regions of the genome with high sequence complexity or low coverage. To address these challenges, methods such as DeepVariant [29] have been developed that utilize deep learning to improve accuracy and sensitivity. Additionally, the use of multiple samples and joint variant calling have been shown to improve variant detection, especially for rare variants. However, the sheer volume of data generated by high-throughput sequencing technologies presents its own challenges, such as scalability and data storage. To address these issues, cloud-based platforms such as Google Genomics, Illumina Dragen on cloud and AWS Genomics have emerged that provide scalable and cost-effective solutions for genomic data analysis. Overall, the state of the art in variant calling continues to evolve as new technologies and algorithms emerge, with the ultimate goal of improving our understanding of the genetic basis of disease and informing precision medicine.

2.3.3 Actual prototype description and maturity level

We have recently defined a prototype variant calling pipeline built around GATK-Parabricks [30] which is currently implemented via Nextflow [31] and that takes advantage of, among other resources, GPUs computing capabilities. In this kind of pipeline, where possible, it is particularly useful to submit jobs to different queues of HPC infrastructures where each process can be associated with a specific queue (for example a GPU queue is desirable for some tasks whereas a CPU one may prove adequate for other activities). This also allows us to easily add new features. In the following we give more details in the current implementation whose maturity can be ascribed to the TRL4 level.

a. Prototype modelization, structure and functional description

In this first prototype we deal with single nucleotide variations detection. Our pipeline is pictorially represented in figure 8. The current implementation is managed via nextflow and the key steps in sequence are: alignment, variant calling, annotation and prioritization. In the variant calling step variants are detected whereas in the annotation and prioritization steps they are biologically characterized and prioritized based on available clinical knowledge on genetic diseases. In the next subsection we give details on the current implementation and each single step.

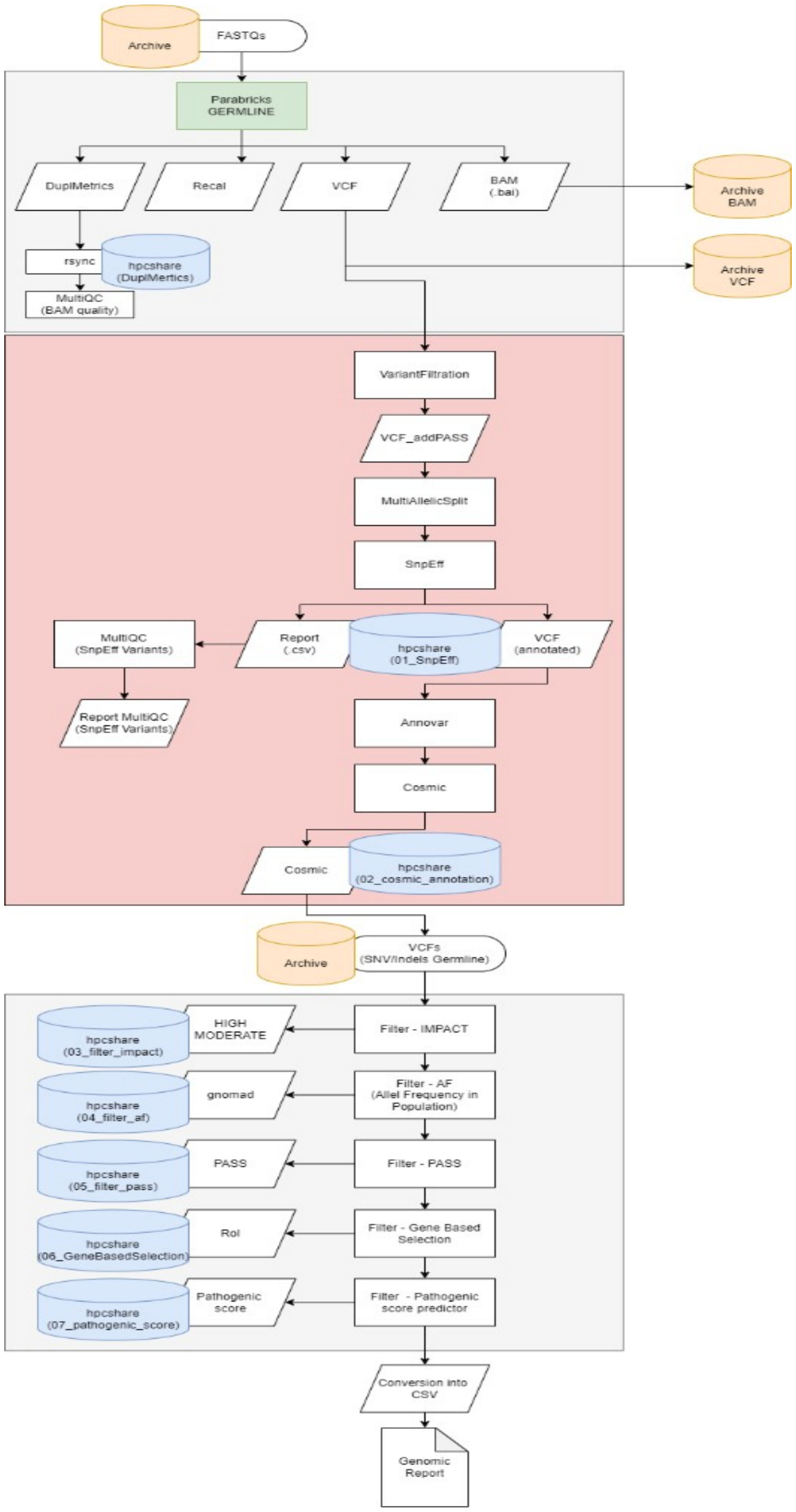


Figure 8: Genomic variant pipeline

b. Actual implementation

Now we discuss in detail the current implementation of the above mentioned three key steps namely: alignment & variant calling, annotation and prioritization.

Alignment & Variant Calling

In the first part of the analysis the nVidia Clara Parabricks package is used. This software is a porting of GATK [28] to a GPUs-based architecture. This package leverages GPUs parallel computing capabilities to accelerate alignment and variant calling. As it is rooted on the "GATK best practices" the produced data are perfectly comparable to traditional pipelines. In detail the function "germline" (the one of current interest) of the package is equivalent to the sequential use of the following tasks: alignment of the DNA reads present in the FASTQ file through BWA-Mem; ordering with respect to the coordinates; mark the duplicates with MarkDuplicates, compute and apply the base quality score with BQSR; lastly perform the variant calling through HaplotypeCaller. During this execution step some files are generated: the BAM file (binary and compressed) contains the information about the aligned reads, the VCF file that contains the identified variants, additionally some auxiliary files are generated which contain the alignment quality (e.g. duplication metrics). Taking advantage of GPUs for a human WGS (at 40x of coverage) Parabricks allows to reduce the analysis time to a few hours whereas on traditional software and CPU-nodes it may easily require 1-2 node computing days. Parabricks is natively containerized hence it is easy to ship it to any GPU-based node. At this step of the execution pipeline, if no further a-priori information is available from the user, one could return the raw data files, namely the BAM and VCF files, one pair for each genome. The total amount of data is about 80 GB for the BAM file and 20 Gb for VCF.

However, at this stage the data is still quite raw and hence further steps are needed to acquire some biological/clinical knowledge. Once the VCF is generated it is possibly necessary to evaluate which variants are possibly false positive. In this case the tool used is VariantFiltration from the GATK package. The generated VCF file (or files) contains only the metrics associated with the reads by the sequencer, but it does not contain any biological information associated with the variant; it only contains the position with respect to the reference genome and the change of base/bases. Hence some further steps are necessary, dubbed annotation steps, that allow including more biological information in the final file.

Annotation

There exist several programs in the scientific Community and among them we choose three largely used ones:

1. SNPEff [32] introduces information relative to the functional effect of the variant and predicts the effect of the variant on genes and proteins (e.g. change of amino acid in the expressed proteins for the case of coding regions).

2. ANNOVAR [33] introduces information about possible modifications that regard the coding region, about possible links with Mendelian diseases, frequencies associated with the variant in the DB such as gnomAD, pathegenocity and conservation scores.
3. In case the sequence is somatic (cancer) the COSMIC [34] tool is used to verify the presence of the variant in its database.

The pipeline produces at this point an HTML report on the data generated by SNPeff through the MultiQC software, which is also used to verify the quality of the output data. Results are hence “published” in specific user selected folders or archived.

Prioritization

Once the annotation step is finished the subsequent step consists in the prioritization of the variants. The prioritization process is characterized by several filtering levels in which the variants, more or less putatively relevant for research and the clinic, are selected. This process is leveraged by the availability of reference databases which support the prioritization process by the explicit knowledge of the correlation between the disease and the variant and hence the genes involved. All the procedures are related to rare variants identification which are the kind of variants that typically correlate with the disease except from multi-factorial more complex cases.

The first filtering step in the prioritization workflow allows to remove the polymorphisms and the common variants present in the VCF, indeed according to the ACMG [35] guidelines those are most probably benign (hence of no clinical interest). Common variant means a variant which has frequency over 5% on the population.

The second filtering level considers the impact that the variant has over the protein expression and it is based on the IMPACT scored assigned by the SNPEff annotator. In this way one can remove all the variants which are synonymous or lie in intronic regions and that don’t have any effect on the splicing.

The third filtering level excludes those variants which cannot be considered to be valid because their quality is not sufficient. This step is necessary to remove false positive which could be ascribed to the reading method of the genomic device. Indeed, devices such as Next Generation Sequencing ones can produce several false positives, particularly where there are repeated sequences patterns.

The fourth filtering level is disease-specific and utilizes a variable list of genes depending on the specific pathologies. For each pathology one has a BED input file, namely a set of genes and positions at which the analysis is carried out.

The last step consists in an evaluation of the pathogenicity scores that again is inspired by ACMG guidelines. First one evaluates the InterVar score [35] and the variant is kept only if the state pathogenic or probably pathogenic is detected. The variants with an InterVar score that is benign or likely benign are discarded. In case the variant bears an uncertain clinical meaning the ClinVar score is evaluated too [36] using a similar acceptance/rejection method.

In the case these two tools lead to an uncertain outcome, the CADD score and the allelic frequency are checked: the variant gets filtered out if the CADD score is below a minimum threshold or if the allelic frequency is more than a given threshold.

c. Validation tests and results

The pipeline has been cross validated with other genomic labs on variants called on real-world input FASTQ files from Illumina NGS devices. A cohort of more than one hundred patients has been used and the results have been compared with the variants reported on a reference WES clinical analysis. Out of the 150 variants reported by the reference WES analysis only 11 were not identified by the proposed pipeline, and of these only two were exonic. A paper is in preparation and the software will be made publicly available.

2.3.4 Prototype evolution and implementation

Currently the prototype maturity level could be ascribed to TRL4 and along the project timeline we can envision targeting TRL5. In the next subsections we deliver details on this process.

a. Prototype evolution direction

The increased TRL can be achieved through an increased flexibility of the software related to both the execution model and the data. Hence the Key Performance Indicators (KPI) here will eventually be two. The first will be the sought-after possibility to remotely run the process on a HPC system, hence along the lines of the so-called Cloud-HPC convergence philosophy; moreover, in general flexibilization of the whole execution process is targeted. The second aim is related to data transfers; currently moving the raw results means possibly transferring several Gigabytes of data rendering the process feasible, yet daunting. Hence the second KPI will be related to the improved data transfer through compression techniques. Further improvements, not currently planned, could be possible and we will consider them along the project execution.

b. Prototype evolution structure and description

While we don’t plan to change the overall backbone of the software, we plan to improve synergistically, or where strictly necessary to replace, the current workflow manager. We plan to insert at the beginning and the end of the workflow some improvements regarding the execution model and the data compression and retrieval. In the next section we report in higher details the involved tools and the related planned improvements.

c. Prototype Implementation and involved tools

Preliminarily to the improvements description it is necessary to deliver some further information on both the data in terms of input and output and code. The number of samples (individuals) will be decided depending on the specific dataset; however each individual translates into two FASTQ.GZ files of size between 10 and 100 GBs. The output of the pipeline per individual is approximately 100 GB where 80 GB ascribed to the BAM file whereas 20 GB related to the VCF. The code is developed in the Nextflow workflow manager language (which runs on top of the Java Virtual Machine) and orchestrates the previously mentioned several tools ranging from python (Parabricks) to the above-mentioned annotation and filtering and tools in various programming languages (e.g. Java, perl). To leverage parallel computing nodes in multi-cores and GPU architectures currently we target the PBS queue based systems.

In this demonstration we first target the adaptation of the current implementation of the pipeline to the **Streamflow** [37] (UNITO, see D4.FL3 section 2.1.6) environment with the aim of increasing the flexibility. We will be employing Streamflow to allow the remote execution of the pipeline and hence render agile the whole execution and to finally retrieve the output data. The increased flexibility will allow us also to test the pipeline in several other (possibly heterogeneous) execution environments hence not only allowing a fast provisioning but also allowing us to evaluate the effect of system hardware/software aspects such the availability of GPUs, different storage and file systems available in the host machines. This will permit us to use the toolset as a benchmark for more or less heterogeneous execution environments where the role of the HW and the software will be evaluated.

The other "tool" we will leverage is data compression. Indeed, currently the pipeline, for each single individual, produces about 80 GB for a single BAM (compressed binary file) and about 20 GB of unfiltered VCF (textual uncompressed file). Moreover, the input FASTQ.GZ files can occupy up to 80 GB of space. In this context, it is interesting to test and possibly propel the further development of file compression tools (**A3lab**, UNIPI, see D4.FL3 section 2.2.4). The compression tools can be applied to VCF files in isolation and groups but possibly even to the input FASTQ files, depending on the data movements involved. We will challenge existing codes (e.g. genozip [38], vcfshark [39]) and possibly combine them with A3lab tools and know-how.

The resulting pipeline will hence endow a higher level of flexibility in the execution and will require less networking (bandwidth) resources thanks to the compression frameworks and retrieval we will be adopting. Further technological investigations may be pursued depending on intermediate results or further ideas that may emerge during the project lifespan.

2.3.5 Final validation tests

As previously mentioned, the two KPI will be the increased flexibility level and the compression ratio. The first KPI will be measured by means of the successful remote execution of the pipeline both in IIT and non-IIT HPC infrastructures (e.g. at UNITO) and also the correct data retrieval. We will be building and delivering pre-configured StreamFlow examples to test on the given systems. Intermediate tests will be carried out at IIT whereas the final ones will be done outside. The success degree of the second aim will be carried through the estimation of the obtained compression ratio compared to the current uncompressed files. For the current experimentation, to avoid data privacy concerns, publicly available data will be employed or in any case data with privacy concerns won't be moved from the original repository.

2.4 Edge-Cloud continuum federation infrastructure

2.4.1 Introduction

The observed evolution of the computing space warranted by networking suggests the emergence of a decentralized, federated, yet seamless organization (see for example [40]). This prediction evokes the concept of the Continuum as a platform infrastructure where data processing may take place dynamically where it is deemed most convenient under any of the criteria of interest to the end user (latency, privacy, energy, etc.). The concept enables the traditional Internet and the Internet of Things to integrate into a seamless Continuum, where a multitude of as-a-service applications may be developed, deployed, and employed regardless of location [41]. The Cloud and the Edge can both benefit from forming the Continuum together, allowing Cloud-like virtualized access to the physical world to occur in a more distributed and dynamic manner, and favoring the creation of numerous novel latency-free, private, and secure, energy-savvy services. This vision of seamless integration extends the literature view, which regards the Cloud and the IoT as distinct spaces, with the letter sending data and offloading computation to the former but not vice versa. Likewise, the concept goes beyond merely connecting network nodes to allow computation to happen at predetermined locations in the computing space. For instance, highly dynamic scenarios comprised of mobile users and diverse applications require flexible placement of data and computing for each mobile user.
The foundation of the Continuum is made up of pervasive service platforms located anywhere the user is, and a multitude of services, with different granularity, available over the Internet and composed opportunistically according to user needs. The concept of Continuum is not entirely novel, as other authors have already described a similar vision in the past [44, 45]. Other researchers, such as [42] and [43], have also proposed analogous visions and architectures for the Continuum, although limited to the innovation of data-driven applications, while [44] suggests adopting the Serverless paradigm [45] for the Continuum. All the mentioned authors fall short in supporting a wide variety of applications, e.g. like long-running industrial control loops or even just mainstream Web app servers. Enabling the Continuum requires design-level (as opposed to merely implementation) considerations as many operational aspects of the system, for example orchestration, are highly sensitive to the specific characteristics of the hosted applications as well as of the hosting infrastructure.
Our effort in this project explored the use of state-of-the-art open-source technologies for building a proof of concept for the infrastructure layer of the Continuum, around the requirements of it being application-agnostic and capable of supporting data locality and computational mobility. Our effort assumed the system concept depicted in figure 9, made of a collection of cluster nodes, each of which allows forming flexible, agile, and geographically bound aggregates of networked computing devices. Each such node federates the resources collectively available within its nodes and orchestrates their deployment. The federation is achieved via a dedicated infrastructure layer, which discovers and aggregates services,

data and compute resources transparently across cluster nodes in a manner that meets end-to-end QoS requirements. As we envision it, the system dynamically instantiates and schedules services along the path from source to destination, based on application-specific requirements and constraints. If a single cluster node lacks hardware, software or data resources to meet the user needs, it will propagate the corresponding requests outside of its federation to cluster nodes within an acceptable geographical distance that have the required capabilities. Collaboration among cluster nodes is essential to support user mobility across neighboring regions. In the Continuum, services should follow the user movements without significant outage or perturbation.

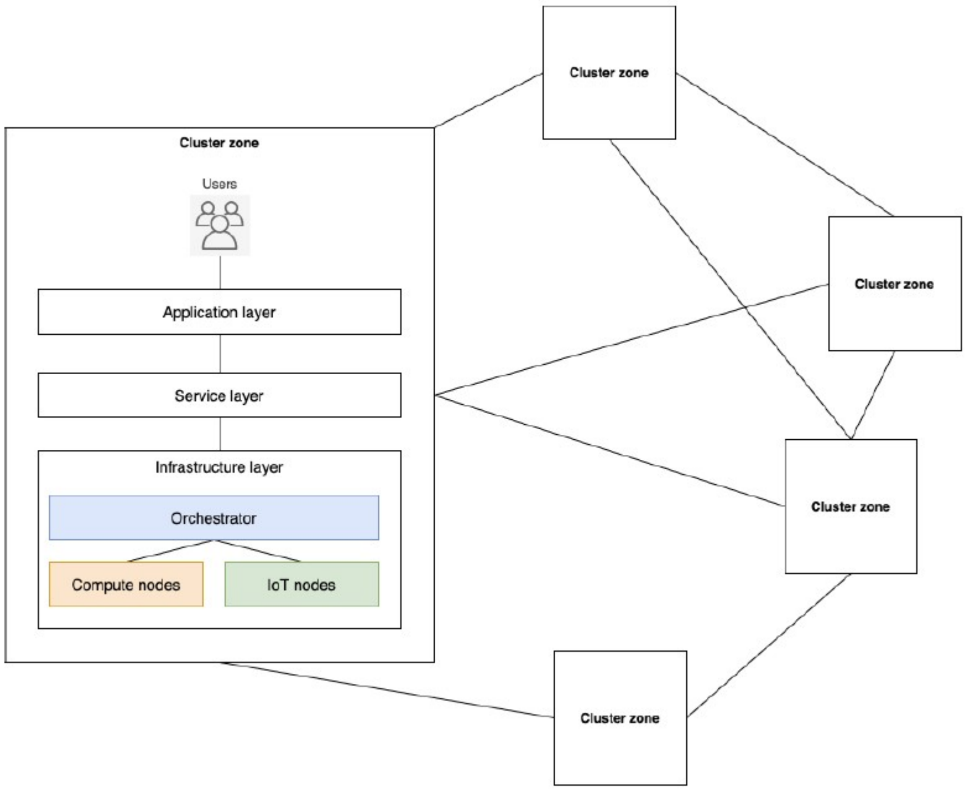


Figure 9: High-level view of a federated set of cluster nodes

User applications running on a single cluster node are given access to requested resources thanks to the intermediation of the service layer. Applications intending to run on a cluster node specify their service requirements and constraints, namely the type of resource (e.g., expected performance, pricing), without needing detailed knowledge of the underlying infrastructure. The orchestrator receives the requirements from the service layer intermediary and provisions resources and services as required, assigning them to compute nodes in the target cluster node. While geographically distant, such nodes form an interconnected cluster that logically aggregates the available resources. Services capture common dependencies like a database and persistent storage for data sources, along with pertinent constraints on them, such as latency limits and subscription plans.
In the exploratory work outlined in this document, we studied the realization of the infrastructure architecture within a single cluster node. Extending the prototype to federations of cluster nodes is left to future work.

2.4.2 Related works

Continuum Computing: The work in [49] provides a comprehensive view of the trend towards integrating Cloud and IoT in a Continuum, and an articulate discussion of architecture, orchestration, privacy and business-value issues. The authors of [50] and [51] offer a broad literature review regarding the integration of Edge and Cloud in a Continuum. Other authors have also proposed architectures for the Continuum, but their efforts were concentrated on supporting data-driven applications [47, 45, 46]. While the magnitude of data produced by the Edge is a major driving factor behind data-driven large-scale workflows, the Continuum vision should extend to a broader range of application types. Besides, while the cited research works achieve some level of integration of Edge and Cloud, they typically fail to consider the need for service composition, uniform interfaces and portable execution throughout the Continuum. Addressing these challenges is crucial to enabling pervasive applications with greater context awareness and mobility. In passing, it should also be noted that the Continuum of Computing is recognized as an emerging paradigm by the HiPEAC (High-Performance Embedded Architecture and Compilation) network of excellence, sponsored by the European Commission [52].

Osmotic Computing: Back in 2014, the authors of [53] described the concept of Fluid Internet. This novel paradigm would seamlessly provision virtualized infrastructure capabilities based on the requirements of services and users, much like a fluid adapting to its surroundings. In a similar chemistry analogy, a few years later, in 2016, the authors of [54] presented the vision for Osmotic Computing. Their work describes a paradigm that enables the automatic deployment of (micro)services composed and interconnected over both edge and cloud infrastructures. Both paradigms present strong affinities to the goals and challenges of the Continuum. Notably, the Osmotic paradigm envisions the same bidirectional flow of microservices from the Cloud to Edge and vice versa, depending on the application configuration. The differences between Osmotic Computing and the Continuum of Computing are subtle but critical in terms of the novelty of the final applications they enable, respectively. First, Osmotic Computing involves deploying microservices, a mere evolution of today's practice of building software in silos. Instead of running the entire application in the Cloud, Osmotic Computing decomposes it into microservices and deploys the latter across cloud and edge datacenters. However, such microservices are not composed of services provided by a ubiquitous intermediary service platform. Osmotic services are thus limited in their context awareness, as services like city sensors are unavailable, decreasing business opportunities. Applications are built instead, at best, in numerous silos [55]. Indeed, the main types of microservices that the osmotic computing framework orchestrates are general-purpose [54]. Second, there is a difference in semantics. Osmotic computing envisions an opportunistic balancing of microservices between the Cloud and the Edge, whereas the Continuum emphasizes a wider continuity in terms of computing. Such continuity spans from the Cloud to the extreme Edge with highly constrained devices, all seamlessly integrated into the Continuum service platform. In contrast, Osmotic

Computing limits itself to comparatively powerful machines such as Raspberry Pi. For such reasons, we emphasize the importance of exploring virtualisation technologies to truly include constrained IoT nodes as active players in the Continuum. Conversely, the Osmotic Computing literature focuses on more resource-demanding container-based approaches. Third, once deployed, the Osmotic microservices are relatively stationary to the deployment location, whereas the Continuum exhibits greater extents of (potential) mobility. In case of unavailability of resources at edge/cloud, Osmotic Computing relies on solutions like message brokers (e.g. Apache Kafka) to store messages temporarily in ad-hoc queues, awaiting to resume services when resources are available [56]. Conversely, the Continuum paradigm expects the computation to migrate to the closest available location temporarily. Additionally, applications in the Continuum can move geographically to accommodate the user's movement, thanks to the seamless and ubiquitous service platform.

Serverless Computing: The Serverless paradigm [45], which focuses on the provision of computational functions, may seem to fit well with the premises of the Continuum. First, the Serverless programming model makes developing, deploying, and managing applications dramatically less burdensome than conventional styles. Second, individual functions may flexibly and equally run on the Edge or the Cloud, thus earning much portability. Furthermore, the current state of technologies we later present, like WebAssembly [46], plays well with the premises of Serverless, with limited resource access. Several works from the research and industry communities are actively exploring the combination of WebAssembly and Serverless Edge functions with notable results [60, 61, 62]. Their work, combined with the definition of serverless work flows described in [44], offer an appealing proposition for the Continuum. However, while well suited for event-driven and request-reply applications, the Serverless computing model falls short for long-running services that must feature high availability and low latency, such as industrial monitoring control loops. Provisioning and instantiating a Serverless function inevitably incur additional latency due to cold start and package download, even more in the face of unpredictable mobile user patterns and distributed networks. Other authors have also proved that it can be challenging to modify stateful applications to the Serverless paradigm, e.g. conventional web servers, since the state is not easily shared among functions [60]. Finally, the Serverless paradigm typically requires limited execution time, limited resource access and limited specialized hardware. While these restrictions allow greater scalability and mobility, they greatly reduce the scope of applications that can be deployed in them. While recent works are reducing such limitations by allowing functions to be quickly co-located in the same machine and memory regions to be safely shared using WebAssembly sandboxing [59], we deem those limitations intrinsic to the nature of the Serverless model. The Continuum and the Serverless models are not to be regarded as one form of computing supplanting the other. Analogously to how the growth of general-purpose container orchestration platforms like Kubernetes was necessary to pave the way for implementing Serverless platforms, we expect a similar direction for the Continuum and the Serverless ways. As the Edge and the Cloud become increasingly integrated, the Serverless paradigm will likely act as the dominant service delivery model within the Continuum.

2.4.3 Actual prototype description and maturity level

Our prototyping effort in this project intentionally used state-of-the-art open-source technologies, in the intent of gauging the distance between the needs of our vision of Continuum and the available technology. The TRL of the prototype is at 3, as it is proper for an experimental proof of concept. The expectation at the end of the project is to have improved critical elements of the selected technology components to close some of the critical gaps, making the infrastructure layer candidate for demonstration in the field, which is proper of TRL 6.

a. Prototype modelization, structure and functional description

As depicted in figure 10, the infrastructure layer of our concept of cluster node comprises a set of service providers that offer data and computation resources. The data can be generated by streaming IoT devices, for example cameras, smartwatches, and other data sources typical of "smart things" environments. The computation resources can be heterogeneous and distributed across the infrastructure, from the Cloud to the Edge. In the following, we briefly discuss each element of the reference architecture for the infrastructure layer.

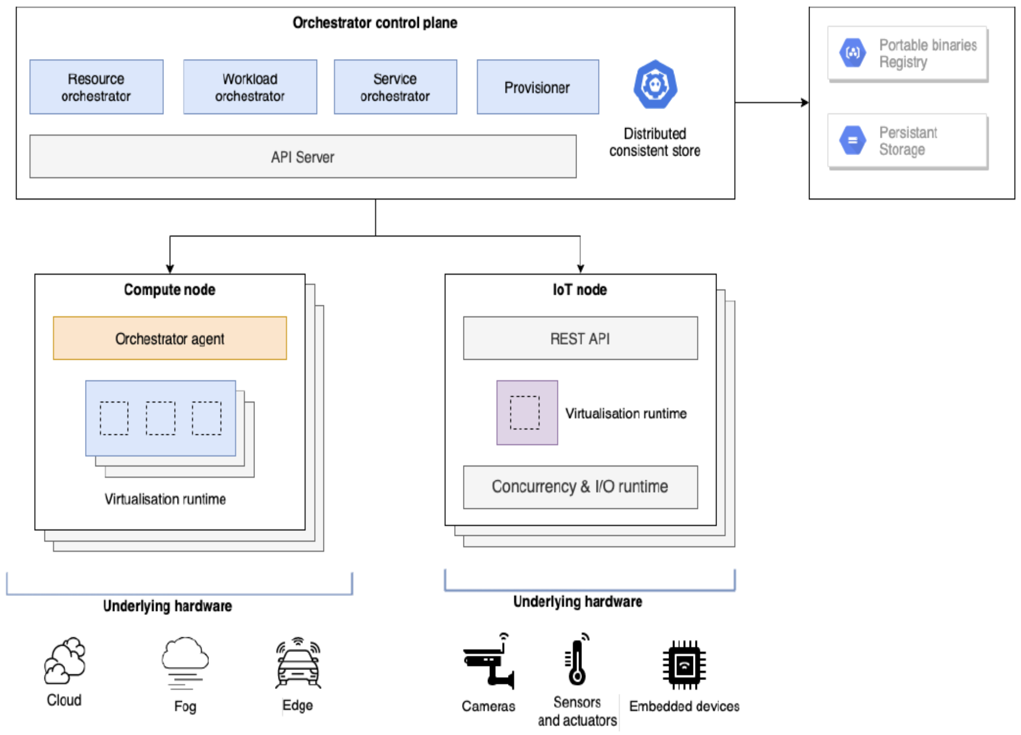


Figure 10: Reference architecture for the infrastructure.

Orchestrator control plane: The orchestrator control plane is the core of the orchestration system. It has a resource monitor module responsible for keeping track of real-time resource consumption metrics for each node in the compute cluster. The scheduler usually accesses this information to make better optimisation decisions. The scheduler is responsible for determining whether the Continuum has enough resources and services to execute the submitted application. If resources are insufficient, applications can be rejected or put on wait until the resources are freed. Another possible solution is to increase the number of cluster nodes to host the incoming application. Such nodes can be provisioned from local machines or anywhere in the network, preferably close to the cluster. After determining if requirements can be satisfied, the scheduler maps application components onto the cluster resources. This deployment is done by considering the application requirements, e.g. latency, geographical constraints, availability or utilization.

Compute nodes: Each machine in the cluster that is available for hosting services and applications is a compute node. Each of these nodes implements the orchestrator agent runtime with various responsibilities. First, it collects local information, such as resource consumption metrics periodically reported to the control plane. Second, it starts and stops service instances and manages local resources via a virtualisation runtime. Finally, it monitors the instances deployed on the node, sending periodic status reports to the control plane. A central responsibility for the virtualisation runtime on the Compute nodes is to offer a consistent execution platform independent of any underlying infrastructure to allow applications to run across all software and hardware types with the same behavior. This capability is a fundamental enabler owing to the extreme heterogeneity of the devices in the Continuum.

IoT nodes: IoT nodes are embedded devices that act as sensors or actuators, provided as services to the cluster (more on this to follow). The IoT nodes are heterogeneous in runtime implementation and communication protocols. Applications in the cluster interface with them via brokers provisioned by the cluster. Besides, the embedded devices support dynamic configuration by running arbitrary virtualisation modules in a lightweight runtime. In addition to warranting interoperability among Compute nodes, the IoT runtime must also be compatible with the application format accepted by Compute nodes, when the module size and the hardware requirements can be satisfied by the target device. Such extended service interoperability enables greater flexibility and novelty in deciding where some aspects of IoT computing, such as controlling and pre-processing, happens. Allowing arbitrary computation to run safely on microcontrollers effectively opens the embedded world to the Continuum as an additional place of intelligent computing, rather than only as a mere data collector and dummy actuator.

Underlying infrastructure: One of the main requirements of the infrastructure architecture is to allow deployment on a large variety of platforms. The cluster machines can be either VMs on public or private Cloud infrastructures, physical machines on a cluster, or even mobile or Edge devices, among others. Such extreme diversity requires rethinking mainstream virtualization technologies in a form that does not require the application programmer to have prior knowledge of the eventual execution contexts.

b. Actual implementation

In this section we outline the implementation choices we made for the realization of our current prototype of cluster node.

Service orientation

The web has become the world's most successful vendor-independent application platform and the dominant architectural style on it is Representational State Transfer (REST) [62] that makes information available as resources identified by URIs. The web is a loosely coupled architecture and applications communicate by exchanging representations of these resources using the HTTP protocol. HTTP is the most popular application protocol on the Internet and the pillar of the Web. However, new communication protocols (e.g. CoAP, which we discuss later in this section) are emerging to extend the web to the Internet of Things and HTTP itself is undergoing revisions (e.g. HTTP/3 or QUIC [63]). Our rationale for picking REST is threefold. First, REST resources are an information abstraction that allows servers to make any information or service available, identified via Uniform Resource Identifiers (URIs). For example, this allows the sensor nodes in our PoC to act as a server and own the resource's original state. The client negotiates and accesses a representation of it. Such representation negotiation is suitable for interoperability, caching, proxying, and redirecting requests and responses. These features enable seamless inter-operation and better availability of any kind of service in the Continuum, especially IoT-involved services. Besides, under the REST architectural paradigm, IoT nodes can advertise web links to other resources creating a distributed discoverable IoT web and resulting in an even more scalable and flexible architecture. Second, REST allows using a uniform interface across the Continuum: clients access server-controlled resources in a request-response fashion using a small set of methods with complementary semantics (GET, PUT, POST, DELETE). The requests are directed to resources that expose a generic interface with standard semantics that intermediaries can interpret. The result is an application that enables layers of transformation and indirection independent of data origin. Third and last, REST enables high-level interoperability between RESTful protocols through proxies or, more generally, intermediaries that behave as server to a client and play as client with respect to another server. REST intermediaries fit well with the assumption that not every device must offer RESTful interfaces directly. Such flexibility suitably accommodates the diversity of communication protocols on the Edge. We used these features to bring IoT nodes into the Continuum as any other service and to enable the coexistence of multiple equivalent services offered by different Cloud providers. We mapped provider-specific interfaces to uniform RESTful interfaces.

Open Service Broker: In our prototype, we realized a web-based service platform that implements the RESTful Open Service Broker (OSB) interface [64]. Components that implement the OSB REST endpoints are referred to as service brokers and can be hosted anywhere the application platform can reach them. Service brokers offer a catalog of services, payment plans and user-facing metadata. The main components of the OSB architecture are depicted in figure 11.

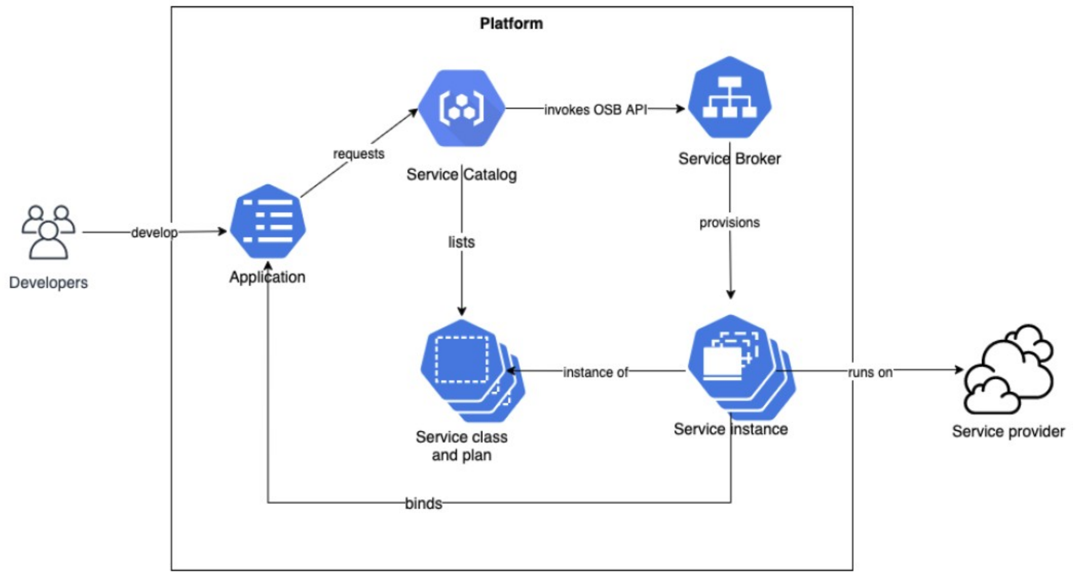


Figure 11: The Open Service Broker Architecture

As Cloud standards still struggle to gain traction, however, we need to bridge the heterogeneity gap between platforms. To this end, we used brokers to orchestrate resources at different levels within a provider. As the number of Cloud vendors is limited, building brokering layers that align access to different Clouds is an affordable endeavor. The service broker translates RESTful requests from the platform to service-specific operations such as creating, updating, deleting, and generating credentials to access the provisioned services from applications. Service brokers can offer as many services and plans as desired. Multiple service brokers can be registered with the service platform so that the final catalog of services is the aggregate of all services. The platform is thus able to provide a rich catalog and a consistent experience for application developers who consume these services. Over the years, the API interface of the OSB has matured considerably, learning from the experience of a wide range of marketplace services and Cloud vendors, such as Microsoft Azure and Huawei Cloud. The current standard version 2.17 is entirely designed around asynchronously provisioned services and provides valuable guidance for challenging situations such as service failures. The OSB guidance ensures consistent semantics and interoperability across various service behaviors. Sadly though, service dependency remains a pain point that needs to be coped with. Currently, the OSB standard does not support a parent-child relationship model between services, whose handling is left inconveniently to the discretion of the broker author. The problems that arise from service dependency include whether to publish multiple services as standalone packages and how to share credentials between services, provision and remove them in the proper order, and solve all these issues uniformly across all platforms.

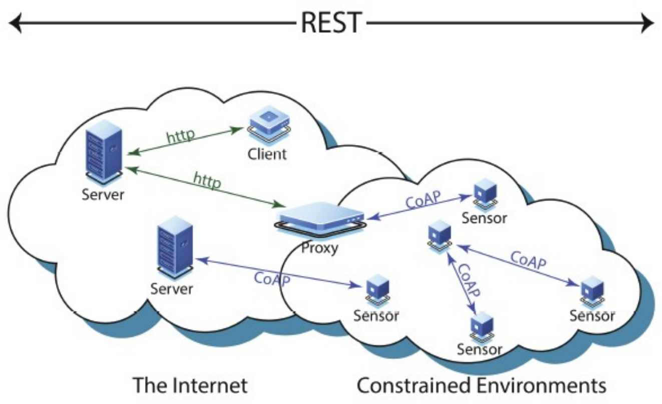


Figure 12: The REST architecture enhanced with CoAP

CoAP: To include IoT nodes in our REST-based architecture concept, we adopted CoAP [65], a web communication protocol for use with constrained nodes and constrained (e.g. low-power, lossy) networks. A central element of CoAP’s reduced complexity compared to HTTP is that it uses the UDP transport protocol instead of TCP and defines a very simple message layer for retransmitting lost packets. The protocol is designed for M2M applications and provides a RESTful architecture between IoT nodes, supporting built-in discovery of resources. As a result, CoAP easily interfaces with HTTP for integration with web services while meeting specialized IoT requirements such as multicast support, very low overhead and simplicity for constrained environments. We made CoAP nodes interoperable with the rest of the Continuum by following the REST architecture's proxy pattern. We built intermediaries that speak CoAP on one side and HTTP on the other without encoding specific application knowledge. Because equivalent methods, response codes, and options are present in HTTP and CoAP protocols, the mapping between them is straightforward. Consequently, the intermediary can discover CoAP resources and make them available at regular HTTP URIs, enabling web services in the Continuum to access CoAP servers transparently in the OSB service platform.

Orchestration

Kubernetes [47]: is an open-source orchestration framework designed to manage containerised workloads on clusters, originated from Google's experience with Cloud services. Two notable features make Kubernetes especially attractive for our PoC. First, thanks to the Container Runtime Interface (CRI) API standardisation, Kubernetes allows for various container runtimes from a technical perspective, with Docker natively supported by the platform. This extensibility allowed us to leverage a uniform virtualisation platform based on **WebAssembly**, while leaving the individual Compute and IoT node to decide the most appropriate runtime (e.g. an interpreter compared to a Just-in-Time or Ahead-of-Time compiler). Second, Kubernetes provides users with a wide range of options for managing their Pods (the most basic unit of deployment in Kubernetes) and how they are scheduled, even allowing for pluggable customised schedulers to be easily integrated into the system. Notably, it also supports label-based constraints for the Pods' deployment. Developers can define their labels to specify identifying attributes of objects that are meaningful and relevant to them but that do not reflect the characteristics or semantics of the system directly. More importantly, labels can also be used to force the scheduler to collocate services that communicate predominantly within the same availability zone, which improves latency very much and paves the way for context-aware services.
Akri: To register the IoT devices on the Kubernetes cluster, we adopted Akri [66], a preliminary Microsoft open-source project which allows visibility to IoT de- vices from applications running within the Kubernetes cluster. Akri stretches Kubernetes' already experimental APIs to implement the discovery of IoT de- vices, with support for the diversity of communication protocols and ephemeral availability. Using Akri, the Kubernetes cluster can carry out dynamic discovery to use new resources as they become available and move away from decommissioned/failed resources. Discovering IoT devices is usually accomplished by scanning all connected communication interfaces and enlisting all locally avail-able resources. Akri is also responsible for enabling applications to communicate with the device and deploying a broker Pod as intermediary. We devised the broker as a web server that abstracts the actual communication between devices and applications behind the RESTful API previously described. Our RESTful broker also helps to scale the number of concurrent HTTP requests by implementing high-performance cache mechanisms. The IoT resource periodically sends its sensor readings to the broker, where the values are cached locally. Each application request is then served directly from this cache without accessing the actual device, with benefits on the average roundtrip time.

As many distributed monitoring applications are usually read-only during their operation (e.g. sensors collecting data in our case), this architecture exhibits great scalability. A potential goal is to enable new types of services where physical sensors can be shared with thousands of users with little impact on latency and data staleness. However, at the time of writing, this is still a very distant achievement. The Kubernetes Device Plugin API heavily influences the current Akri architecture. Such interface, already considered experimental by the Kubernetes community, was designed for hardware attached to compute nodes, e.g. GPUs. However, IoT devices can live independently from the nodes, and most of them do. Akri expects a 1:1 relationship between compute node and device, whereas most IoT devices do not have any kind of relationship to any node per se. This mismatch has several undesired consequences, especially on scalability and resiliency. Another pain point in Akri's current state is that the project lacks more advanced yet very needed features for implementing software caching or assuring high availability or autoscaling in IoT scenarios. Such features are admittedly harder to provide but highly needed to bring the Cloud to the Edge and vice versa, an essential preliminary step to the Continuum.

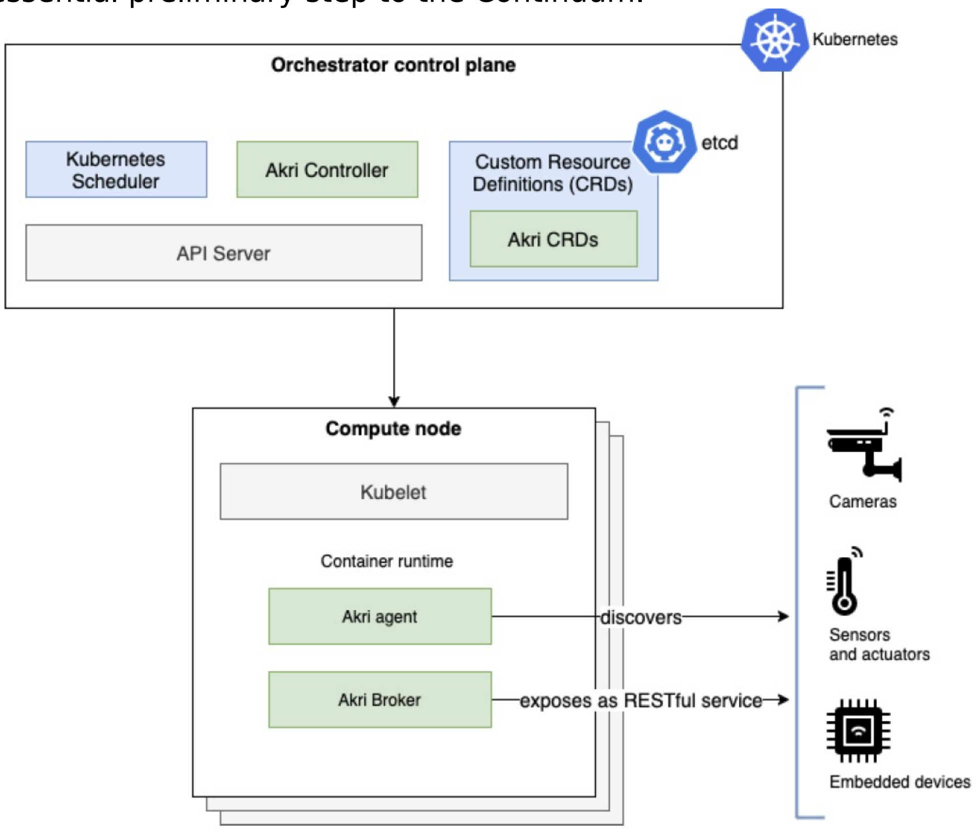


Figure 13: The Akri contribution to the infrastructure layer

Figure 13 shows that the Akri architecture can be divided into four main components: the agents, the controller, the brokers and the configuration. A configuration extends the Kubernetes API with new communication protocols and the related metadata, such as the protocol discovery parameters or the Docker image for the agent container. The Akri agent is a Pod responsible for discovering devices according to a communication protocol. It keeps track of the device state and communicates status updates with the Akri controller.

Virtualisation, interoperability and portability

WebAssembly (Wasm) [48], first announced in 2015 and released as a Minimum Viable Product in 2017, is a nascent technology that provides strong memory isolation (through sandboxing) at near-native performance with a much smaller memory footprint. WebAssembly is a language designed to address safe, fast, portable low-level code on the web. Developers who wish to leverage WebAssembly may write their code in a higher-level (compared to bytecode) language such as C++ or Rust [67] and compile it into a portable binary that runs on a stack-based virtual machine. We picked WebAssembly as the technology enabling virtualisation, interoperability and portability in the Continuum for two fundamental reasons. First, WebAssembly provides language-, hardware-, and platform-independence by offering a consistent execution platform independent of any underlying infrastructure to allow applications to run across all software and hardware types with the same behaviour. The importance of such a feature for the Continuum cannot be emphasised enough. Second, WebAssembly is advertised as safe and fast to execute. No program code can corrupt its execution environment, jump to arbitrary locations, or perform other undefined behaviour (which memory-safe languages, such as Rust, contribute to preventing). Thanks to that execution guarantee, a WebAssembly may suffer only data exploits mitigated by applying memory and state encapsulation at the module level rather than the application level. Granular memory encapsulation means that even untrusted modules can be safely executed in the same address space as other code, a critical point for dynamic configuration in constrained devices and multitenancy in the Compute nodes of our architecture. Performance wise, benchmarks of Wasm runtimes on modern browsers have shown a slowdown of approximately 10% compared to native execution, typically within 2x [49, 51]. WebAssembly is currently looked at as a candidate method for running portable applications without containers. Ideally, WebAssembly can provide significantly more lightweight isolation than VMs and containers for multi-tenant service execution. This idea is still in its infancy, but there has been consistent interest around it in recent years ([57], [58] and [59]), especially for serverless computing.

Another strong point of WebAssembly is enabling arbitrary code execution on highly constrained devices across the Continuum. The authors of [68] and [69] have also explored various WebAssembly-based mechanisms for safe arbitrary execution on constrained devices and have evaluated the trade-offs between efficient Wasm processing and memory consumption. Generally speaking, Just-in-Time compilers for WebAssembly exist (e.g. Wasmtime [70]) and receive more attention from the community, but their size and complexity make them unsuitable as yet for microcontrollers. Although WebAssembly interpreters can often run more than 10x slower than native C [71], they help dynamically update and debug system code, but are not yet mature in terms of performance and energy efficiency. Interpreting WebAssembly on microcontrollers offers an appealing alternative to other language runtimes. The WebAssembly standard has many features that make it attractive for embedded devices [69]. First, as mentioned before, WebAssembly can be generated from different source languages and run on many CPU architectures. Furthermore, many broadly used language runtimes such as JavaScript, Lua, or Python cannot provide predictable execution. They may require excessive memory for a microcontroller, whereas Wasm requires no mandatory garbage collection and only a few runtime features around maintaining memory sandboxing. This lightweightness is a most valuable asset in an embedded adaptation.

Overall view

Figure 14 summarizes the key technologies employed in the prototyping of the infrastructure layer of our reference architecture for the Continuum.

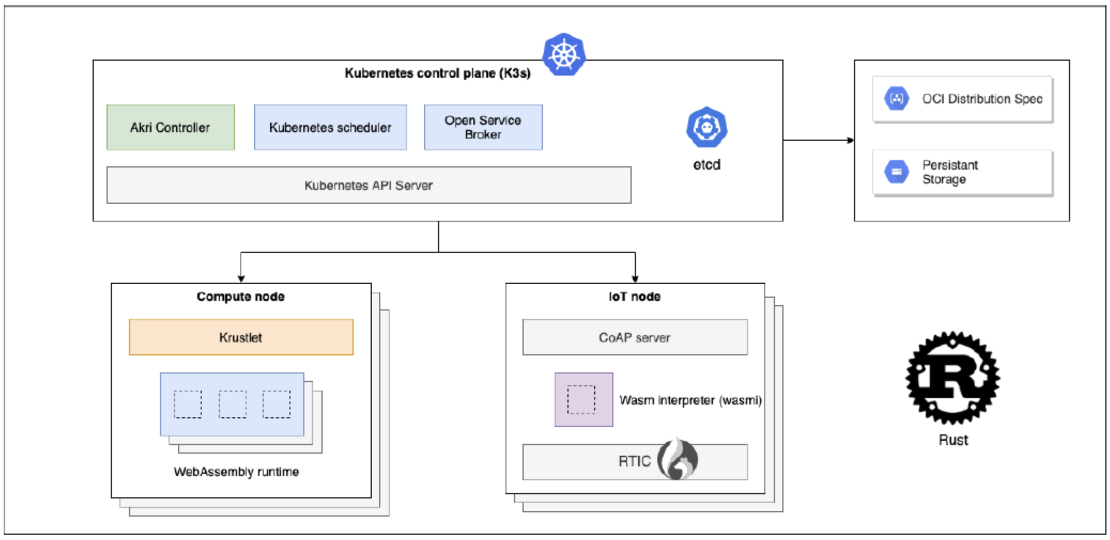


Figure 14: Technology baseline for the architecture of the reference infrastructure

c. Validation tests and results

Assessing the fitness of WebAssembly for the Continuum infrastructure

Arguably, WebAssembly is one central enabler in our quest for the Continuum, for portability and virtualization, but also for computational mobility. On that account, we performed an in-depth analysis of how WebAssembly fares technically in our prospective implementation. We evaluated WebAssembly's fitness for Continuum purposes along three axes.

1. Its suitability as a portable binary format for pure computational services, showing its significantly smaller size than other mainstream alternatives unfit for highly constrained IoT devices.
2. Its fitness as an interpreter for the same embedded devices to understand whether it can guarantee safe virtualized execution while keeping the promise of reasonable performance and predictability.

3. Its maturity for Cloud-like capabilities. We present a cluster of Kubernetes nodes whose virtualisation runtime is based on a Just-In-Time Wasm compiler. The nodes accept applications distributed as Wasm container images under the Open Containers Initiative Artifacts specification [72].

For our evaluation, we have used the following devices:

- **Edge cluster nodes**: 4 Raspberry Pi 4 Model 3B+ with Quad-core Cortex- A53 (ARMv8) 64-bit SoC at 1.4GHz and 1 GB physical memory. The Raspberry 3B+ model has been chosen to showcase the feasibility of the presented technologies on limited low-powered machines, relatively cheap and with only 1GB of memory. Our results are comparable with other research regarding containerised virtualization over Raspberry Pi [61];
- **Sensor nodes**: STM32F407 microcontrollers with ARM Cortex-M4 core, 512KiB flash storage, and 128KiB of memory. The device is also capable of many 32-bit floating-point operations. Raspberry Pi and STM32F407 microcontrollers are designed for moderately high computational performance, low unit cost, and power efficiency in Edge and IoT computing environments.

We trust these empirical results generalize for ARM machines and microcontrollers in the Cortex-M family.

Axis 1: Wasm for IoT devices

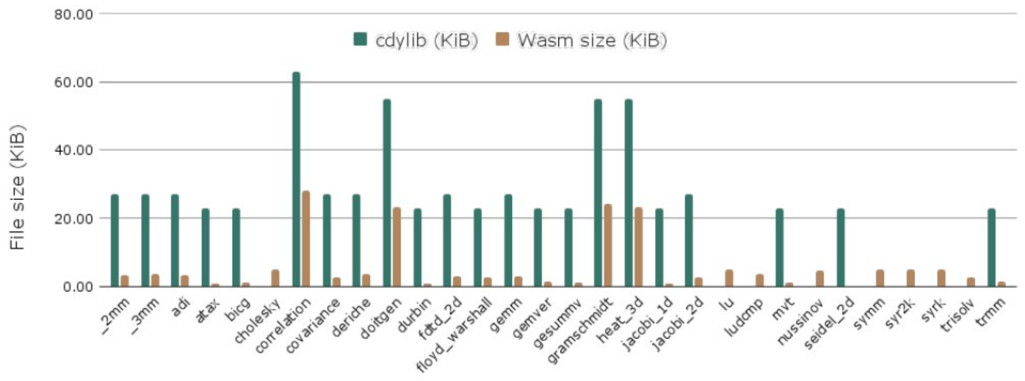


Figura 15:Wasm size vs C dynamic library size (in KiB).

Figure 15 compares the sizes of different Wasm binaries compiled from the Polybench [73] modules. The Polybench benchmark suite offers relevant functions to embedded systems as it includes standard matrix and statistical operations. We have chosen the C dynamic library size as a meaningful comparison since it is a close alternative to Wasm binary files. Both outputs have been compiled from the same Rust source code and using the same LLVM toolchain and optimisation flags. The results undeniably favor the Wasm binary format as the C dynamic lib is often many times larger. Comparing Wasm files to containers would be even less relevant and greatly favor the former, as containers package a whole operative system filesystem, which is unnecessary for pure computational IoT services. Even the tiniest image base (Alpine Linux Mini Root Filesystem) has an additional size of about 5.5MB uncompressed.

Axis 2: Wasm as an interpreter

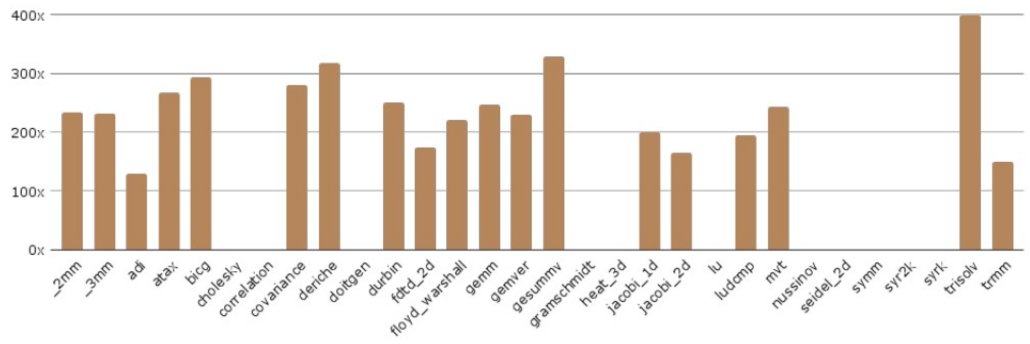


Figure 16: Wasm interpreter vs Rust native performance.

Figure 16 plots the slowdown of the Wasm interpreter executing Polybench benchmarks on the STM32F407 microcontroller against native Rust. The results show a dramatic slowdown, with a reduction factor of 100-400x. Such results dispel the prospect of using Wasm interpreters on microcontrollers to support dynamic reconfiguration. However, the Wasm interpreter we used, wasmi [74], was the only available Rust WebAssembly interpreter, and we adapted it to work on embedded devices. The interpreter was designed for safe execution in the blockchain instead of efficiency in highly constrained devices. Alternative embeddable interpreters, implemented in the memory-unsafe C language, show a much inferior execution penalty, in the order of 30-60x slower than native [69]. Arguably, however, a 30x execution penalty can still seriously deter the usage of interpreters in microcontrollers.

Another crucial concern we have found in our work is that the heap overhead of using Wasm interpreters is not predictable. Such unpredictability does not come again in favor of the usage of WebAssembly on microcontrollers, as embedded devices have extremely limited resources and must have predictable behaviors to ensure proper execution. Such deficiency is an intrinsic issue with interpreters, as the code instructions and execution data structures must be stored in heap memory. This behavior contrasts with the binary executables that can save and access instructions or read-only data on the more capable flash storage. Writable data is saved in the stack instead, and it can be estimated with accuracy in many production-grade toolchains like C and Ada.

Finally, running Wasm on resource-constrained microcontrollers also presents a memory-design issue. Wasm’s pages are 64KiB by standard, too large for microcontrollers that often have between 16-256 KiB RAM. Dynamic allocation is a common requirement even for embedded systems. However, Wasm specifies that the sandbox should expand memory by 64KiB chunks, insufficiently granular for constrained embedded systems. Consequently, we had to adapt the interpreter to allocate non-standard pages of 16KiB. Otherwise, it would have been impossible to execute any benchmark on the STM32F407 microcontroller, as additional heap space is required for the interpreter’s internal structures and the Wasm instructions themselves.

Wasm on the Cloud

We successfully integrated WebAssembly into Kubernetes with Wasm Pods running on Krustlet [75], while the container Pods are scheduled on

K3s [76] Kubelet. Krustlet (a Kubernetes-Rust-Kubelet stack) is an experimental implementation of the Kubernetes compute node (Kubelet) API that supports Wasm as virtualisation technology. Therefore, it listens to the Kubernetes API event stream for new units of execution (Pods) and runs them under a WebAssembly System Interface (WASI) runtime (notably, Mozilla's Wasmtime [70]). K3s is a fully certified Kubernetes distribution geared towards Edge environments backed by a commercial company. K3s is implemented in Go and packaged as a single binary of about 50MB in size. At the time of writing, it has yet to be possible to implement a portable web server and compile it as an application to Wasm. There is an underlying issue with implementing network servers as there is neither sufficient network API nor multi-threading support in the standard yet. On the one hand, the current WebAssembly System Interface (WASI) standard only contains a few methods for working with sockets that are insufficient for complete networking support. Adding support for connecting to sockets is fundamental to allow Wasm modules to connect to web servers, databases, or any service. On the other hand, the lack of concurrency primitives means that a server running in WebAssembly is single-threaded, or its implementation has to be significantly more complex (e.g., like Node.js's event loop [77]). This limitation severely restricts the workload capabilities of the server. Lately, the WebAssembly specifications have outlined a thread and atomics proposal intending to speed up multi-threaded applications. At the time of this writing, however, that proposal is still in the early stage and implemented only in web browsers, behind an experimental flag.

The hands-on evaluation presented in this paper has taught two main lessons. On the one hand, that, in concept, the WebAssembly technology is especially fit for the Continuum as we envision it. On the other hand, however, the current standard and the corresponding industrial implementations significantly penalize resource-constrained platforms, making them an exceedingly inadequate destination for applications seeking temporary hosting as part of compute mobility. The underlying theme is that the Wasm specifications (notably memory management, networking and concurrency) are not mature or robust enough as yet for use with real-world applications, let alone for innovative Continuum services. At the time of writing, navigating the WebAssembly landscape to figure out how to create ad-hoc workarounds to such limitations [78], remains a distinct (and annoying) responsibility of the application developer, which also makes portability harder to achieve.

2.4.4 Prototype evolution and implementation

a. Prototype evolution direction

As noted earlier, we reckon the TRL of our current prototype implementation to be at 3, as proper of an experimental proof of concept. Our target at the end of the project is to get as close as possible to TRL 6, which we expect to achieve by improving critical elements of the selected technology components, thereby making the infrastructure layer candidate for in-the-field demonstration in a yet-to-be-determined application domain.

b. Prototype evolution structure and description

Our architecture concept assumes user applications to consist of the execution of workflows that traverse service components deployed dynamically across compute-capable nodes of the Edge-to-Cloud Continuum. This vision gives rise to the notion of "dynamic orchestration", where the execution workflow is specified in terms of required services and the matching to provided services may be resolved dynamically based on user-preferences, service levels, privacy, energy, and latency requirements. One direct implication of that vision is that the deployment of service components selected for the workflow may be dynamic and opportunistic, and therefore may also contemplate mobility and migration. Figure 17 depicts this notion.

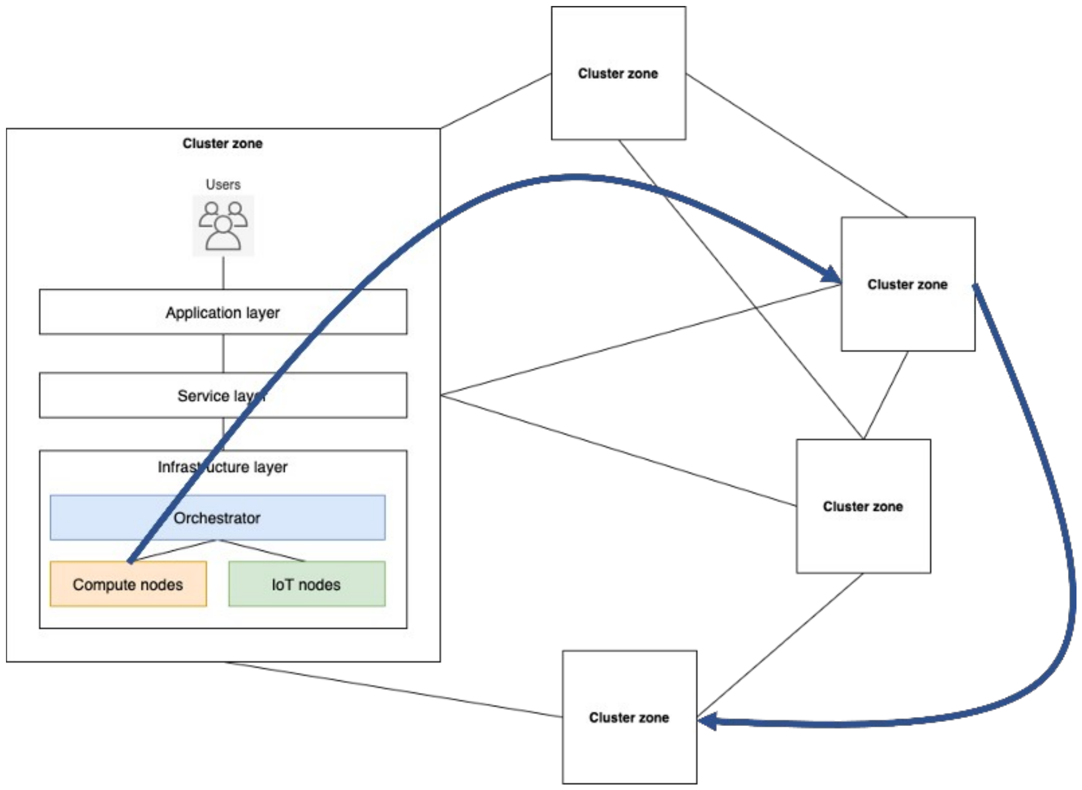


Figure 17: example routes of mobility of "compute bundles" across federated cluster zones.

c. Prototype Implementation and involved tools

Enacting the vision of the previous section, entails several distinguishing requirements, which reflect the discussions in that section and that we recall now in this conclusion, in a bottom-up fashion.

1. The first significant requirement is that the service components of interest to our concept should be realized as "compute bundles" that can be deployed dynamically in safe sandboxed "envelopes" hosted in resource-adequate Edge nodes. We plan to meet this requirement with an enhanced version of the **WebAssembly** runtime, which we want to improve for the time predictability of its execution behaviour, and for its "embeddability", that is to say, for its goodness of fit for functional capabilities coupled with frugal footprint for compute and storage needs.

2. Allowing compute bundles to migrate requires understanding they may have ongoing communications with client endpoints, which may also share a connection state. This scenario requires server-side connection migration, which is not currently implemented in the QUIC standard, but may be supported by **MoveQUIC** prototype proposed in [128] and extensively described in deliverable report D4.FL3, section 2.4.2.
3. The WebAssembly runtime would be the central element of the Compute node and should be able to negotiate CoAP-level API bindings to nearby IoT nodes for sensing and actuation actions on controlled devices. To that end, the Compute node should feature local orchestrator capabilities, which we plan to provide with an improved **Krustlet**-to-Wasm integration, where "improved" means disentangled from Kubernetes and capable of supporting deployments within and outside of the boundaries of its own cluster zone. Krustlet is especially interesting to us as it is implemented in Rust, a language that we consider especially fit for embedding in predictability-aware system aggregates. Figure 18 depicts the internal architecture of a cluster zone in our architecture concept.

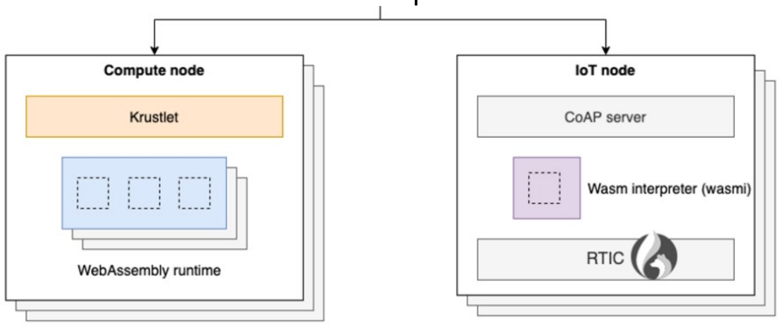


Figure 18: an inside view of a cluster zone made of one Compute node paired with an IoT node.

4. The next level up in our system concept is the orchestrator control plane across federations of Compute nodes. In that direction, we shall continue to explore the usability of Akri, which is designed for cooperation with a Kubernetes-type scheduler, and we would like to extend to non-Kubernetes control planes. In parallel to that, we shall look at ways to incorporate a single cluster zone in larger federations, using **Liqo.io** (extensively described in deliverable report D4.FL3, section 2.3.3).
5. Describing and deploying the user application as a dynamic orchestration of an execution workflow will need a flexible orchestration platform for which we plan to explore the use of the **INDIGO** orchestrator, the third integration-candidate technology described in deliverable report D4.FL3, section 2.3.2.

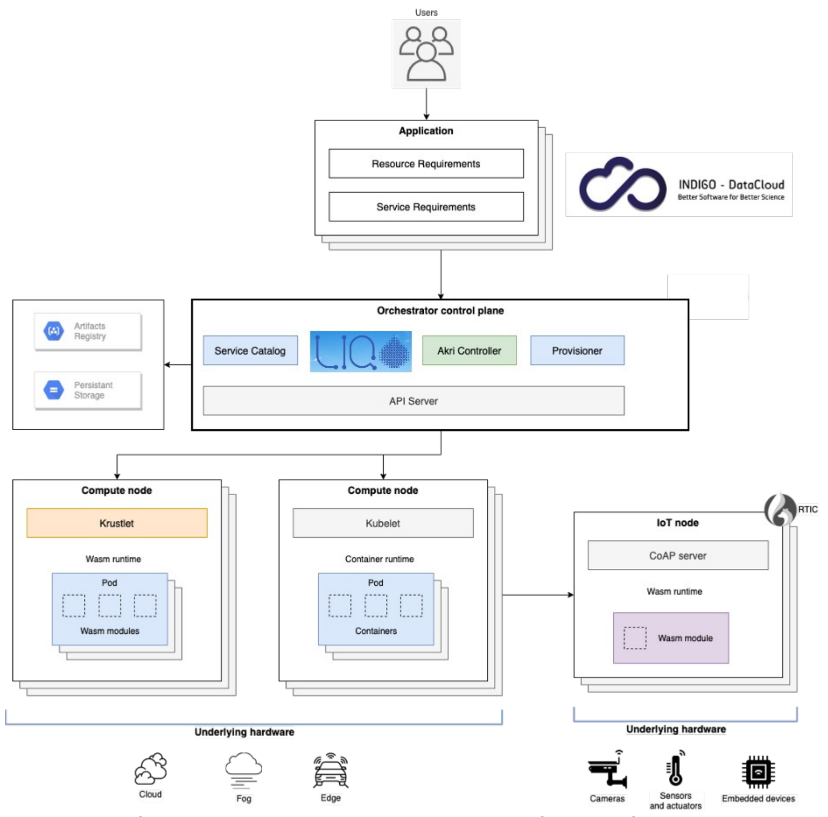


Figure 19: overall view of the proposed system concept, with identification of the key technology candidates.

Figure 19 provides a pictorial representation of the overarching system concept described in this section, which positions the key technologies that we plan to employ in its prototype realization. The resulting use case will need to be put to trial against an application scenario of industrial interest. What this application may be remains to be determined at present, but discussions about it are ongoing with national and international partners.

2.4.5 Final validation tests

The validation tests that will be needed to evaluate the soundness, viability and performance of this vision include several scenarios, each of which evoked by the previous enumeration:

1. We shall test our WebAssembly implementation and see how better is performs than the standard version in scenarios similar to those discussed in section 2.4.4.
2. We shall test how well the MoveQUIC technology integrated in our use case serves the migration needs of compute bundles at the server-side of the user application. The primary axis of evaluation for these tests will be continuity of service during “live” migration, that is, without interruptions of service.
3. We shall test the ability of our Kruslet variant implementation to operate outside of Kubernetes and to support deployments within and outside of the boundaries of its own cluster zone.
4. We shall test how Liqo.io fits in the federation-level orchestration plane foreseen by our architecture concept, and how well it integrates with Akri for interfacing with edge resources.
5. We shall test how adaptable the INDIGO orchestrator is to the concept of dynamic orchestration entailed in our system vision.

2.5 Interactive Computing Service

2.5.1 Introduction

The Interactive Computing Service (IAC) provides access to computational data on an HPC system via a web interface, with near-immediate access to such resources. This is an alternative approach to the traditional access to HPC resources, which is usually based on ssh access on a remote login system on which a scheduling system dispatches the jobs on some queues. There are multiple reasons why such kind of approach might be preferable with respect to the “traditional one”:

- Avoiding queues might be preferable for small amount of resources.
- Workflow can be modified interactively while is running, with respect to intermediate results.
- A web interface is more user-friendly, extensible, and allow the use of visualization tools.
- Web sessions can be easily restored when closed.

In our approach the base interface we choose is Jupyterlab [91], extended and customized with respect to our needs. In particular, our aim is to adapt the “classic” Jupyterlab approach (which is web-based) onto the computational nodes of an HPC center (which are optimized for parallel computations rather than act as web servers). These two different components traditionally head in opposite directions from the technological point of view, thus In order to match them we will employ several software tools:

- a frontend part which is exposed to the web.
- a backend part which run a collection of software to be employed by the user for his/her parallel computations.
- a server/client approach in the middle in order to fill the gap between the two components above.

The frontend part is the one exposed to the web, and also manages user authentication and resources selection (see below). The backend part is thought to rely on the HPC infrastructure, and the software which is provided to the user is organized in different releases, in order to maintain retrocompatibility when upgrades are released. The server/client approach exploits the Slurm scheduler and the BatchSpawner feature of Jupyterlab (properly extended to our needs).
Once the user access to the framework, a form is displayed, to be filled in order to specify for instance:

- the amount of requested resources
- a time limit for the session
- which backend release he/she wants to use

After that, a web interface with all the software contained in the chosen backend release is shown to the user; the software is running on the HPC nodes and can be freely employed by the users for their purpose.

2.5.2 Related works

E4 Engineering [100] is the main developer of the framework from which this infrastructure is derived; the initial setup for the implementation described so far is derived from their GAIA suite [101], in particular from the version developed in the context of the ICEI project [102] of the Fenix Research Infrastructure [103].
Most of the work developed in the ICEI context has been to deploy the core part of the services, starting from the open source solutions available for the main components needed: the core part of the infrastructure relies on Jupyter [130], which handles the web interface once the server is allocated on the backend part: this operation is handled via Slurm [131], and the joining link between these two elements is represented (in our case) by a modified version of the Batchspawner implementation [132]. The work related to the Fenix Research Infrastructure was to provide a general deployment tool for all the supercomputing centers which are project partners (CINECA, BSC, CEA, CSCS, JSC), and the solution provided by E4 has been built with Ansible playbooks [92] available for all the involved partners; the prototype solution has been deployed at CINECA using both the cloud infrastructure “ADA cloud” [133] and some dedicated compute nodes from Galileo 100 cluster [134], and the result constitutes the development infrastructure described in the sections below and the starting point for the work that will be realised in the project.

2.5.3 Actual prototype description and maturity level

At the present stage it can be estimated a TRL level of 4, since a pre-production and a development platform are up and running at CINECA; the latter has allowed testing of experimental features before the pre-production scale.

a. Prototype modelization, structure and functional description

So far, the IAC framework in the development environment works smoothly with a frontend interface running on the HPC cloud infrastructure hosted at Cineca (ADA Cloud), relying on a local restricted user database and with a dedicated Slurm controller running in the same cloud environment, in order to grant flexibility in terms of testing the scheduler configuration. On the backend side, two HPC nodes have been isolated and dedicated to such a framework; such nodes are perfectly equivalent to the ones that will be used at production scale.

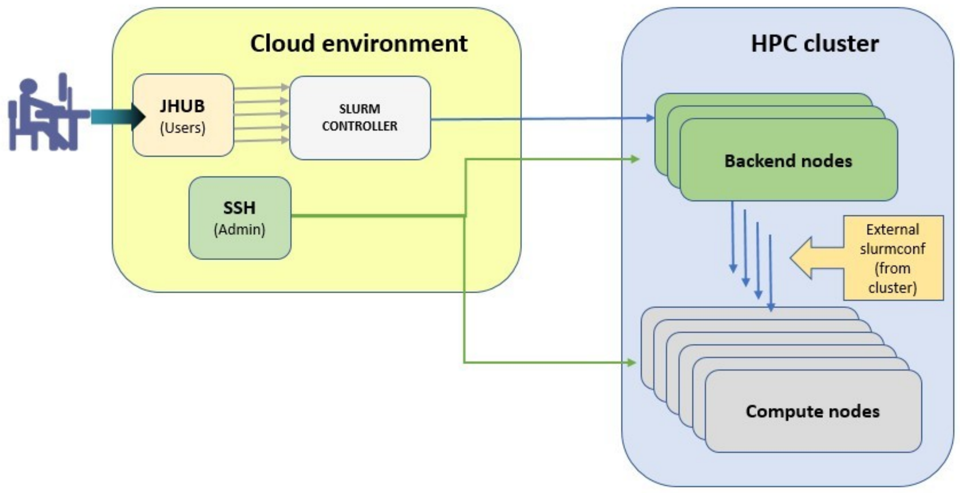


Figure 20: Interactive Computing Service architecture

b. Actual implementation

The code is mostly composed by Ansible playbooks [92] and Python scripts, relying on Conda [93] and Mamba [94] installers; everything relies on Jupyter as main dependency as well as Jupyter Server Proxy [95] for spawning different servers with respect to the traditional Jupyter experience; Slurm scheduler is the last essential component for the whole framework. Many additional dependencies are present but optional, based on the additional extensions and tools (e.g. Xeus kernel [96] for C++, Julia [97] and R [98] kernels, VSCode interface [99] that need to be deployed.

c. Validation tests and results

Several tests have been collected to be used as test cases and validation procedures for the current implementation; in particular, tests involving GPU utilization have been deeply tested using all the modules which support GPU. Such tests have been collected in an internal git repo and can be made publicly available. Customized kernels and integration with the HPC module environment using them have also been tested. More in general, any workflow involving the environments made available by the service can be used as a valid test case to validate the infrastructure.

2.5.4 Prototype evolution and implementation

The development setup described so far resembles the same setup of the production environment, including all its features but with the possibility to tune them with additional degrees of freedom.

For instance, the dedicated Slurm controller allows to test ad-hoc scheduler parameters without the risk of compromising the jobs running on the cluster. In the same fashion there might be multiple instances of the frontend interface, e.g. running on different virtual machines, relying on the same backend infrastructure. The same approach can be easily obtained at backend level: each collection of tools displayed in the launcher is packed in a single “release”, and different releases can be added and chosen in the login phase by the user; since backend releases are completely independent one to each other, ad-hoc releases can be added with different purposes and content.

a. Prototype evolution direction

We aim to further extend the current infrastructure exploring different additional tools: for instance, a remote desktop implementation is still missing and it will be tested for instance via TurboVNC [104] and/or Xpra [105] integration; R-Studio [106] and Octave [107] will be also tested for integration.
Another important direction we would like to explore is the deployment of the framework on the bare metal nodes for the incoming Leonardo infrastructure [108], which hosts a set of visualization nodes which are worth exploring as a possible hosting environment for the IAC tool.
Further needs by other partners involved in the project will be explored, trying to meet their needs to the fullest possible extent.

b. Prototype evolution structure and description

The production environment would look exactly the same as the development one, besides the fact that the frontend part is relying on the compute nodes and the Slurm controller of the production environment; there’s no relevant difference in the frontend setup, which can be hosted either on bare metal nodes or once again in the cloud environment, like in the scheme in figure 21.

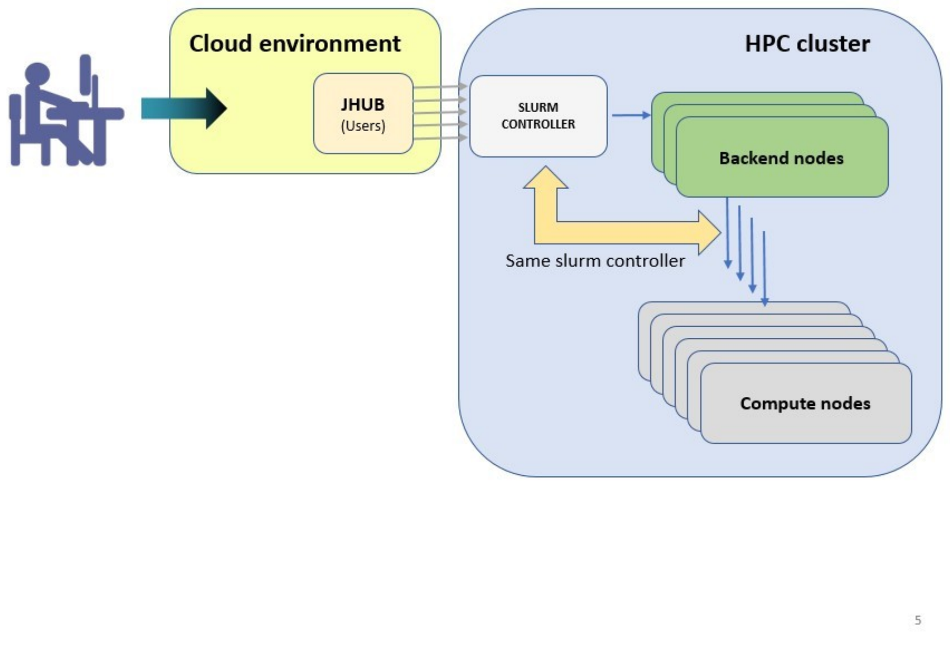


Figure 21: evolution of the Interactive Computing Service architecture

In the end, this general approach allows us to test different combinations of frontends and backends when novel configurations will be needed in the context of the CN-HPC: this makes it possible to try to address the needs of all the partners of the CN-HPC, fitting with other tools in this proposal.

c. Prototype Implementation and involved tools

A valuable synergy and cooperation will hold with the **Jupyter Workflow** tool and the **StreamFlow** tools presented by UNITO in D4.FL3, respectively in section 2.1.5 and section 2.1.6; Jupyter is the cornerstone of the IAC implementation; StreamFlow implementation by UniTO can be tested in order to try to operate a distributed workflow among different platform via the Interactive Computing interface; this approach would be highly innovative, since it could co-operate with the interactive computing interface on two different levels:

- on local basis, distributing the Jupyter cells execution using compute nodes interactively in place of backend nodes; this could largely improve the resources availability for interactive workflows, spanning the execution of the single Jupyter cells on the whole cluster.
- on a remote basis, since it would make available all the cluster and the cloud options available for the same user logged to the service.

Another integration that will be explored is with the **VisIVO** tool presented by INAF here in section 2.2; a Python wrapper for the VisIVO cli will be developed in order to ease the interaction with the Jupyter notebooks; at this point it will be possible to create in the Interactive computing interface a custom environment for the visualization and data analysis using VisIVO and make it available for the users via web browser; in addition, when the VNC integration will be implemented in the IAC, the VisIVO graphical user interface can be tested and eventually used directly from the browser.

Furthermore, we can integrate the **DivExplorer** tool presented in D4.FL3 section 2.2.9 by PoliTO inside the Interactive Computing framework. DivExplorer offers a Python library to be executed in a local environment: the modules contained in such library can be easily included in an ad-hoc kernel to be displayed in the Jupyter launcher of the Interactive Computing interface. The kernel would be available to all the users who have access to the service, and easily accessible from the web browser; the users could also benefit from the monitoring tools included in the IAC in order to identify and fix bottlenecks in real-time during the notebook execution.
In terms of deployment, the integration between the tools would require including an Ansible playbook for the ad-hoc environment, with a compliant approach with respect to the ones already present for the deployment of the service.
On the DivExplorer repository are also hosted several use cases [129] in form of Jupyter notebooks, which can be used as validation tests for the integration of the tools.

2.5.5 Final validation tests

The same tests reported for the development infrastructure can be also used to validate the production one. Such dataset could be integrated with some tests in order to validate the integration with VisIVO, to be added in collaboration with the INAF team; the above-mentioned tests for the DivExplorer integration [129] can be added too to the set of validation tests.
Summing up, the Key Performance indicator to be achieved at the end of the project are the followings:

- **Toolset expansion**: in order to enhance the user experience using the IAC service the aim is to increase the number and the variety of tools that are offered to the user; in particular this KPI aims to add at least three of the following tools in the default launcher:
    - Remote desktop visualization (via VNC and/or Xpra protocols)
    - Julia kernel
    - R-Studio web interface
    - Octave web interface
- **Distributed workflow**: Integration with Streamflow and Jupyter workflow tools presented by UniTO has the aim to run Jupyter cells from IAC implementation on a remote host. This KPI can be considered achieved if the integration among these tools is successful, in particular if:
    - Jupyter Workflow and Stream Flow are successfully implemented inside a IAC development implementation
    - Such implementation will be able to run the above-mentioned validation tests on a remote host (e.g. a remote cloud environment and/or an HPC cluster)
- **Integration with DivExplorer**: To achieve such integration with the tool presented by PoliTO, a "DivExplorer" environment has to be

successfully added in the launcher of a IAC implementation, and in such environment validation tests for DivExplorer has to be successfully run

- **Integration with VisIVO**: To achieve such integration with the tool presented by INAF in section 2.2, a “VisIVO” environment has to be successfully added in the launcher of a IAC implementation, and in such environment validation tests for VisIVO has to be successfully run
- **Implementation on Leonardo**: In order to achieve this KPI, a IAC implementation has to be successfully run on Leonardo cluster[108], and validated with the above-mentioned validation tests.

2.6 Serverledge: QoS-Aware Function-as-a-Service in the Edge-Cloud Continuum

2.6.1 Introduction

The Function-as-a-Service (FaaS) paradigm emerged as an evolution of Cloud computing services, relieving users from infrastructure management and resource allocation responsibilities. It allows users to deploy fine-grained functions, developed using their programming language of choice, and execute them in a serverless fashion.

A lot of effort has been spent recently trying to provide the benefits of FaaS to applications running at the edge of the network (e.g., Internet-of-Things applications). However, the limited availability and heterogeneity of computing resources at the edge, as well as the challenges of geographical distribution, call for specific architectures and policies for FaaS at the edge of the network. First solutions have been proposed for Edge-based FaaS, including light function sandboxing techniques instead of OS-level virtualization. However, these solutions either work within single Edge nodes or scale over multiple nodes without considering geographical distribution. Therefore, we lack a platform with the ability to span both Edge and Cloud and adaptively exploit both.

Serverledge, a FaaS framework designed for the Edge-to-Cloud continuum at the University of Rome Tor Vergata, aims to fill such a gap. Serverledge adopts a decentralized architecture, with nodes organized into edge zones and cloud regions based on their location. Every Serverledge node, being it at the edge or in the cloud, is able to schedule and execute invocation requests with minimal or no interaction with the rest of the system, keeping latency as low as possible. To cope with load peaks and extend Serverledge node’s local capacity, Serverledge also supports vertical (i.e., from edge to cloud) and horizontal (i.e., among Edge nodes) computation offloading, allowing nodes to forward invocation requests that cannot be served locally. Our framework accounts for differentiated groups of users, each characterized by one or more Quality-of-Service (QoS) requirements, possibly specified in terms of response time, availability, energy consumption.

Serverledge is implemented in Go, supports functions written in multiple programming languages (specifically, Python, JS, and any language through custom images), currently relying on simple-yet-popular Docker containers for isolated function execution.

We have designed Serverledge with flexibility in mind, aiming to contribute a flexible and easy-to-extend prototype to the research community, for future investigations on FaaS at the Edge. To this end, Serverledge is an open-source project available on Github [121] and has received the artifact badges at IEEE Percom 2023, where Serverledge was first presented.

In this project, we plan to extend Serverledge following multiple directions, that span from the runtime management layer to the virtualization/containerization layer and providing different mechanisms and policies that include function offloading and migration and energy- and QoS-aware scheduling policies.

2.6.2 Related works

Existing open-source FaaS frameworks (such as OpenFaaS [109] and OpenWhisk [110]) are not well suited for Edge environments, mostly because of: 1) the use of centralized schedulers or gateway components, which introduce latency in geo-distributed settings, 2) memory-demanding function sandboxes, usually based on software containers, 3) and overly simple and best-effort scheduling policies, which do not account for the complexity of Edge infrastructures. Therefore, researchers started investigating solutions to better support FaaS at the Edge and novel frameworks have been recently presented that better suit Edge environments. They often exploit lightweight function sandboxing mechanisms instead of OS-level virtualization (e.g., Faasm [111] and Sledge [112, 113]). However, these solutions either work within single Edge nodes or scale over multiple nodes without considering geographical distribution.
The solutions closest to Serverledge are Faasm [111] and Colony [114], as they support function execution offloading. Faasm [111] is an open-source research prototype that introduced Faaslets, an isolation abstraction for high-performance serverless computing. Relying on Faaslets, Faasm significantly reduces the initialization time and memory footprint of function sandboxes, compared to container-based approaches. Moreover, Faasm has built-in support for function chaining and state management. Faasm runs using multiple worker nodes, which can schedule and offload requests horizontally to other workers. However, Faasm does not explicitly consider geographical distribution of the nodes.
Colony [114] is a framework for parallel FaaS in the Cloud-Edge continuum. Its goal is to let nodes process data on their resources while also offering their computing capacity to the rest of the infrastructure. Colony differs from most existing FaaS frameworks as it relies on task-based programming models through COMPSs. The generated workflows are then executed over the infrastructure, possibly offloading tasks both horizontally and vertically.
Sledge [112, 113] and tinyFaaS [115] are other FaaS frameworks specifically designed for Edge environments, aiming to provide serverless execution with reduced resource consumption. The key difference between the solutions mentioned above, including Serverledge, and these two frameworks lies in the fact that Sledge and tinyFaaS target single-node deployment scenarios and, thus, they lack the ability to exploit Cloud resources.
Other works (e.g., [116, 117, 118]) study architectures and algorithms for function placement and load distribution in decentralized FaaS systems, but relying on the existing Cloud-oriented frameworks for actual function execution, possibly incurring the issues mentioned above when running at the Edge.

2.6.3 Actual prototype description and maturity level

Serverledge is a decentralized FaaS platform designed for Edge-Cloud computing environments. Serverledge allows users to define functions through high-level programming languages and automatically allocates resources for their execution upon invocation. Following the approach adopted by most the existing FaaS platforms, Serverledge currently executes functions within software containers, which are spawned as needed and initialized with the code and libraries required by each function.

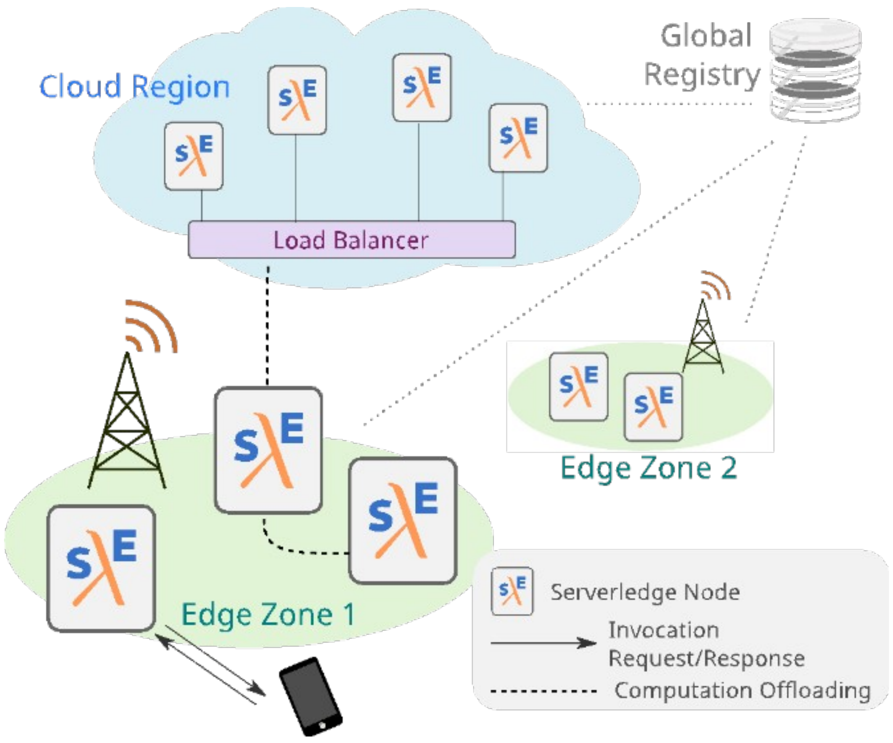


Figure 22: Overview of Serverledge architecture.

Figure 22 illustrates the high-level architecture of a Serverledge installation, which consists of one or more nodes deployed either in Cloud data centers or at the edge of the network, and a global registry. The latter provides distributed nodes with the required data about the system, including membership information about the deployed nodes. Within the registry, nodes are organized into different cloud regions and edge zones based on their location. Cloud regions typically represent geo-distributed data centers, while edge zones may be associated with, e.g., single towns or cities. Each cloud region may further comprise a load balancer to distribute incoming requests to the nodes deployed in the region. Note that, while the global registry represents a single logical entity in the architecture, it may be deployed with multiple replicas for scalability and fault tolerance.

The core idea underpinning the design of Serverledge is that there are no single or privileged entry points for function invocation. Indeed, users can send invocation requests to any node (e.g., one in their proximity).

Compared to FaaS platforms designed for the Cloud, scheduling functionalities are not centralized and, thus, every node is able to schedule the execution of incoming requests. This is particularly important for Edge-generated requests, which are not forced to reach a centralized gateway in the Cloud for scheduling.

Serverledge adopts a per-request container scaling behavior, where new containers are only spawned when needed. In particular, when an invocation request enters the system, if enough resources (i.e., CPU and memory) are available, a new container is spawned and initialized to execute the function. When this happens, the request has to wait for the container to be fully initialized before being served and it experiences a cold start. To reduce cold start frequency, containers are not immediately destroyed after function completion and are kept in a warm pool until a fixed timeout expires. If one or more warm containers are available, these can be re-used to serve new requests for the same function, thus avoiding a cold start.

Because of the limited resource capacity of Edge nodes, it is likely that a single node (and perhaps a whole edge zone) cannot sustain the incoming load. Therefore, Serverledge allows nodes to offload invocation requests to other nodes, when needed. In particular, we support both vertical and horizontal offloading. The former refers to execution requests being forwarded from edge to cloud nodes, whereas the latter indicates request offloading among edge nodes.

We now analyze in more detail the architecture of a Serverledge node, whose architecture is illustrated in figure 23. Each Serverledge node comprises the following components: *API Server*, *Scheduler*, *Local Registry*, *Offloader* and *Container Pool*.

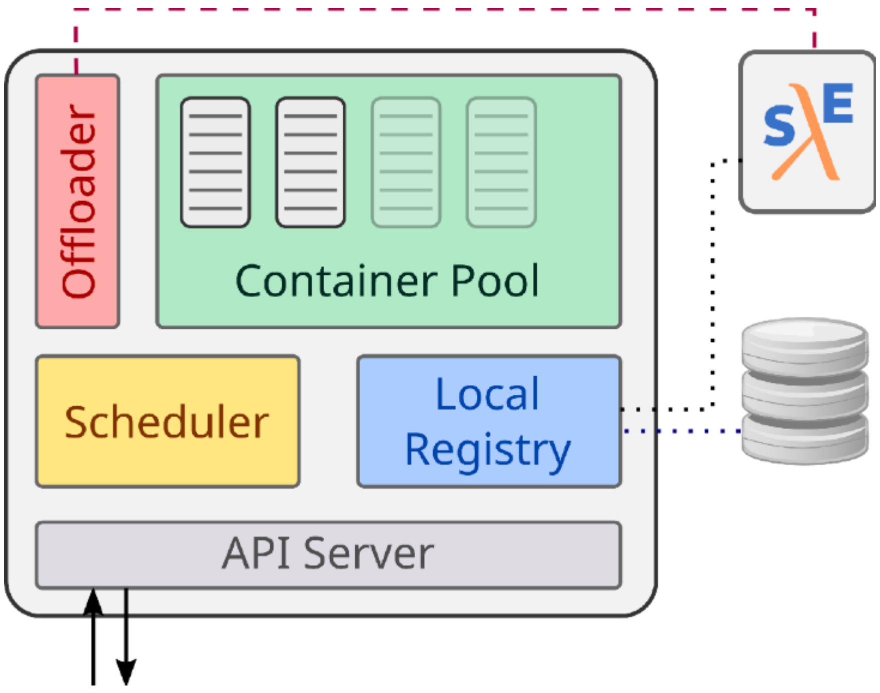


Figure 23: Architecture of a Serverledge node.

The API Server provides a set of key functionalities through an HTTP API, which is primarily used by client applications (e.g., to create and invoke their own serverless functions), but it is also accessible to other Serverledge nodes (e.g., for offloading). In particular, each node supports the following key operations:

- **/create** to register a new serverless function;
- **/invoke**: to invoke an existing function, possibly specifying one or more input parameters and QoS requirements for the submitted request;
- **/list**: to get a list of the registered functions;
- **/delete**: to de-register an existing function;
- **/status**: to obtain information about a node (e.g., amount of available resources and current state of its container pool).

Serverledge uses a Global Registry (see figure 22) to store information about the nodes in the system and the registered functions. To provide low-latency access to the shared information stored in the Global Registry, each Serverledge node is equipped with a Local Registry, which acts as a local cache. Besides caching, the Local Registry stores local and neighborhood information (e.g., who are the neighbor edge nodes and their amount of available resources) that is collected and managed on the node itself, without propagating it at global level. To build and update such neighborhood information in an efficient and scalable manner, each Serverledge node relies on the gossiping-based Vivaldi algorithm, which is traditionally employed in peer-to-peer networks.
When a function invocation request is received from a node, it is passed to the Scheduler, which provisions the required resources for execution if possible. If local execution is either not possible because of resource lack or shortage or not convenient in order to fulfill the request QoS requirements, the Scheduler can either decide to offload the request to another Serverledge node or to drop it. Figure 23 illustrates the different decisions that the Scheduler can take for each function invocation request.

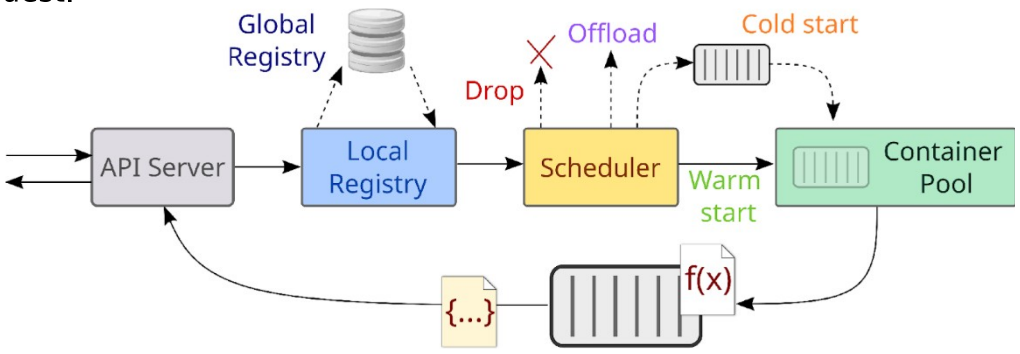


Figure 24: Function scheduling decision: if local execution is not possible, the Scheduler can drop the request or offload it.

Let's now consider the case in which the Scheduler serves locally the request on the basis of the scheduling policy decision. If the request requires a new container, it is spawned and initialized by copying the source code package of the function into the container. Since containers are not immediately destroyed after function completion and are instead kept in a pool of warm containers, they can be reused to serve new requests for the same function, thus avoiding a cold start. Besides picking or creating containers for function execution, the Scheduler can apply other decisions to incoming requests. First, the Scheduler can decide to offload requests to another node, which will take care of actual function execution. Offloading decisions can be driven by the limited resource capacity of edge nodes, due to which a single node (and perhaps a whole edge zone) cannot sustain the incoming load. Furthermore, requests can

be dropped by the Scheduler and, thus, not executed at all. As regards the offloading policy, Serverledge currently provides a proof-of-concept naïve policy that also integrates QoS support. According to it, the Scheduler tries to process each request locally on the node. If this is not possible, because there is not enough memory on the node, the request is offloaded. Specifically, latency-sensitive requests are offloaded to a neighbor edge node, to avoid the additional network delay. Conversely, best effort requests are offloaded to the Cloud, to preserve edge nodes.
The Offloader is in charge of supporting the mechanism for both vertical (i.e., from the edge to the cloud) and horizontal (i.e., within an edge zone) offloading. When the Scheduler makes an offloading decision for a request, a target node is selected relying on the Local Registry, which provides information on the neighbor edge nodes and the available cloud regions (if any). Then, the request is forwarded to the selected node. According to the offloading mechanism currently supported, the local node acts as a reverse proxy, submitting the invocation request to the API of the remote node. The local node waits for the computation result traveling back from the remote node and sends it back to the invoking client as soon as possible.
Serverledge performance has been evaluated experimentally against three state-of-the-art FaaS platforms, which are OpenWhisk, tinyFaaS and Faasm. Compared to Apache OpenWhisk, in an Edge-like deployment, Serverledge shows a dramatically higher throughput. Compared to tinyFaaS, Serverledge has a slightly lower throughput. However, compared to tinyFaaS, Serverledge can scale the execution beyond a single node through offloading and increase the overall throughput. Compared to Faasm, Serverledge shows comparable response times, and even better on average; benefits of Faasm are still evident looking at the maximum response time because of the lightweight function runtime it employs. More details on the evaluation, as well as other details regarding the implementation, are discussed in [124], that is integrated by a companion artifact paper at PerCom 2023, which has received two badges (Artifact Certified and Result Certified). In the light of these experiments, we can classify the TRL of our prototype as 3, that is Experimental proof of concept.

2.6.4 Prototype evolution and implementation

Serverledge has been designed with the aim to fill the gap between Edge and Cloud and provides a flexible and extensible framework for FaaS in geographically distributed environments. While Serverledge provides a suitable framework for low-latency FaaS execution in the Edge-Cloud Continuum, several challenges must still be addressed to fully support QoS-aware execution and scheduling in such a dynamic environment. We analyze how Serverledge can evolve following multiple directions, that span from the runtime management layer to the virtualization/containerization layer.
Starting from the orchestration layer, Serverledge can be extended to support *function composition and state management*, enabling the execution of complex applications. Serverledge currently supports the execution of single functions and the orchestration of more complex applications, likely composed of multiple functions, is under the responsibility of the invoking clients. We plan to allow users to define their applications as workflows by composing functions. To this end, a serverless workflow language such AWS Step Functions or AFCL [119] can be

supported by Serverledge. The latter is a YAML-based language that supports a rich set of constructs to express advanced control flow and data flow. Deploying and executing workflows clearly requires a revision of the scheduling and offloading policies in use.

Implementing stateful applications on top of serverless functions, which are stateless by nature because of their ephemeral execution environments, is challenging. While a naïve solution relies on third-party data stores to externalize application state, this approach leads to poor performance due to the additional latency incurred every time the state is read or written. Therefore, it is necessary to design strategies and mechanisms to manage application state alongside functions.

As regards the runtime management layer, *live function migration* can be provided in Serverledge. While serverless functions usually have a short duration, long-running functions are gaining popularity as approaches for serverless data analytics and machine learning are explored. The existence of such workloads at the edge calls for live migration mechanisms, so as to possibly migrate running function instances to free up resources as needed or, in general, to respond to adaptation needs. Indeed, function migration to a different node can allow the system to revise initial scheduling decisions that become far from optimal over time, to reschedule a resource-consuming and long-running function on a different node having more powerful resources, or to support smooth movement of mobile users during function execution. To the best of our knowledge, live migration has been exploited very limitedly for serverless functions. Needless to say, the development into Serverledge of function migration paves the way for the design of *migration policies* and their integration with offloading policies in a comprehensive manner. In addition, the support for migration is strictly related to the virtualization/containerization techniques used in Serverledge.

Serverledge nodes can offload incoming invocation requests to neighbor edge nodes or remote cloud nodes. However, offloading decisions must be carefully planned by *offloading policies* on the basis of a multitude of factors, including the desired QoS of the requesting user, the resource demand of the invoked function, the current load of the node and the network. As a global formulation of the problem for the whole system would not scale for realistic deployments, we envision the development of decentralized offloading policies.

Finally, at the orchestration layer *load balancing policies* among the Serverledge nodes belonging to the same zone can also be studied, taking into account the availability of warm containers in order to avoid the cold start.

At the virtualization/containerization layer, Serverledge currently executes functions within Docker containers. Other *containerization engines* (e.g., Podman), as well as *microVM* (e.g., Firecracker) can be integrated within Serverledge. The integration of lighter *function sandboxing techniques*, such as WebAssembly-based runtime environments, is another direction to pursue at this layer. *WebAssembly* (Wasm) has been already proposed (e.g., in Faasm [59] and Sledge [57, 113]) as an alternative method for running serverless applications at near-native performance, while providing strong memory isolation, small memory footprint and optimized invocation time. In particular, Serverledge can leverage WasmEdge to accelerate the serverless functions. WasmEdge [122] is a lightweight, high-performance and extensible WebAssembly runtime for cloud-native, edge and decentralized applications. Finally, another possible direction to explore regards the support of even lighter and specialized function

execution environments, such as those provided by unikernels. To this end, a promising open-source project is Unikraft [123], which reduces virtual machine and container image sizes to a few KBs by tailoring the operating system, libraries and configuration to the particular needs of the application.
At the infrastructure layer, we currently do not select and scale (horizontally or vertically) the nodes on which Serverledge will be deployed. Serverledge can thus be integrated with *placement and auto-scaling policies* that take place at the infrastructure layer.
At the hardware layer, Serverledge can be extended to run on *specialized* or *resource-constrained devices*. This extension is intertwined with the support of lighter virtualization techniques that allow functions execution on heterogeneous resources.
As regards the QoS parameters, Serverledge currently provides the ability to specify performance-related metrics, such as the function response time. The monetary cost can be easily added to consider the FaaS execution on edge nodes which are managed by third-party entities or on on-demand Cloud resources. However, to increase the *energy awareness*, we call for a holistic effort in this direction in the context of Serverledge. Indeed, considering the energy consumption of computer systems is increasingly important for environmental and economic aspects. Energy awareness is especially important at the edge of the network, where computing devices may be equipped with limited energy resources (e.g., battery-powered sensors or smartphones). A FaaS system comprising such energy-constrained nodes should necessarily take scheduling decisions in an energy-aware manner, so as to extend the device lifespan and reduce the need for frequent battery replacements or recharging (e.g., offloading requests with less tight latency requirements). While researchers have started considering energy aspects within FaaS systems (e.g., [120]), there are not yet established techniques and tools to measure the energy footprint of serverless functions across different implementations and deployments. Indeed, measuring the energy consumption of applications presents general challenges which we inherit, along with FaaS-specific issues that mainly arise from the FaaS programming model. Specifically, an open issue is how to measure the per-function energy footprint and whether estimates that rely on models turn out to be sufficiently accurate for our scope. Moreover, it is also interesting to develop new Serverledge system components able to predict the energy required to execute functions as well as to perform lower-level tasks (e.g., initializing a container), in order to devise proactive energy management approaches.

Traditional *KPIs* for edge systems, and thus FaaS systems that operate in the cloud-edge continuum, include response time and throughput. In our evaluation of Serverledge, we have relied on these KPIs, using for the workload different functions either taken from existing FaaS benchmarks (e.g., Sieve and Fibonacci sequence) or developed on purpose (e.g., a binary image classifier based on a convolutional neural network).
Besides traditional KPIs for serverless edge systems, we also plan to consider power-efficiency related KPIs.

As discussed in section 2.6.3, we reckon the TRL of Serverledge current prototype to be at 3, which has been also validated during the artifact evaluation that took place at IEEE Percom 2023. Our target at the end of the project is to reach TRL 6. We expect to reach this level by improving core components and features of the Serverledge architecture as discussed above.

We intend to integrate Serverledge with the *energy efficient orchestration and resource management in the cloud continuum* tool provided by the University of Pisa (**UNIPI**, see D5.FL3 section 2.3.4) with the overall goal of providing a holistic energy-efficient management that spans from FaaS execution at the edge to FaaS frameworks deployed in Cloud data centers.
UNIPI already experimented with energy efficient orchestration and resource management at the cloud level. Given the FaaS timing constraints, the goal is to minimize the energy consumption. This is achieved by managing how FaaS resources are redirected to different cloud nodes and by taking into account the current load of each node and trying to consolidate the allocation of resources to power off some of the nodes whenever possible. The current tool will be extended and integrated with the Serverledge toolkit to manage heterogeneous nodes from both the cloud and the edge. It will also handle vertical offloading from the edge to the cloud and viceversa, whenever the status of the underlying resources or the evolution of the application requirements need that.
Other tools that we would like to investigate include **MOVEQuic** (**UNIPI**, see D4.FL3 section 2.4.1) for function live migration and the support of WebAssembly described by the University of Padova within their prototype.

2.6.5 Final validation tests

The KPIs used to evaluate the success of our tests mainly rely on response time and throughput, which can be easily measured using open-source monitoring tools (e.g., Prometheus,[125]). To evaluate the performance of lightweight runtime environments for FaaS as alternatives to Docker containers, specific KPIs are the cold start latency and the memory consumption.
If energy-awareness is integrated into Serverledge, we will also need to identify proper energy measurement tools at the hardware and software levels. As regards the latter, the PowerAPI middleware toolkit (https://powerapi.org) allows us to estimate the power consumption of applications without the need of deploying physical power meters. This toolkit, which relies on SmartWatts software power meter, could be exploited to obtain the energy footprint of functions.
As regards the workload for the final evaluation tests, a proper mix of functions (and serverless workflows, if supported) from existing (and upcoming) serverless benchmarks should be carefully selected in order to test the different features, mechanisms and policies provided by Serverledge. A proper mix of functions should include CPU-intensive and memory-intensive functions, as well as stateless and stateful functions, possibly developed in multiple programming languages.
To generate the load, the Locust tool (https://locust.io) can be effectively used. It is a Python-based load testing tool that allows us to emulate the behavior of concurrent users issuing function requests, with configurable think times or maximum rates. Locust supports running load tests which are distributed over multiple machines and can therefore be used to simulate millions of simultaneous requests and evaluate Serverledge scalability.

2.7 Improving I/O phases in computational modelling of Galaxy Formation

2.7.1 Introduction

The formation and evolution of galaxies and of the Supermassive Black Holes (hereafter SMBHs) at their centres is a central theme of contemporary Astrophysics and Cosmology. Numerical modelling of this problem has proven to be challenging due to the truly long-range nature of the gravitational interaction, which cannot be shielded [136]. For these reasons the *computational complexity* of algorithms devised to model galaxy formation and evolution poses challenging problems when coded in parallel codes. Very often astrophysical codes are adopted as *testbeds* of new hardware architectures, as they are able to challenge their scaling capabilities.
The prototype we are proposing is designed to cope with a well-known state-of-the-art parallel code, **FLASH** [135]. This code implements a spatial and temporal partition based on a *Adaptive Mesh Refinement* (hereafter AMR) decomposition and a rather sophisticated hierarchical tree schemes to deal with the long-range gravitational interactions. Our main aim consists in improving the frequent I/O phases using tools made allowable within the current FL3, in particular two of them: **Nethuns** and **CAPIO** .

2.7.2 Related works

The I/O of large checkpoint and generated data files from large numerical runs executed by large, parallel codes represents a serious bottleneck in many codes. These issues have been discussed for the **FLASH** code, and few solutions have been proposed [137, 138]. However, the performance of these approaches in terms of scalability up to exascale computing platforms has not yet been demonstrated.

2.7.3 Actual prototype description and maturity level

We will now provide a short description of the two prototypes we are going to use.

FLASH [140] is a modular AMR Computational Fluid Dynamics (hereafter CFD) code. It was originally developed to model thermonuclear flashes occurring inside stars which will develop into Type II Supernovae, and in particular the detailed evolution of the radiative and thermal balance during the expansion phases. The AMR structure of **FLASH** proved to be particularly effective in numerical modelling in general highly inhomogeneous systems: for this reason, the code was enriched with a large amount of physical modules to model phenomena like e.g. radiative cooling, magnetic fields, cosmic ray transport, etc. Besides that, also the range of available numerical solvers has been significantly extended, and as of today includes *PPM, Split, Unsplit Hydro Solver*, *Riemann Solver*, and few more.

Besides these, a *Relativistic Hydro Solver*, a *Magnetohydrodynamic Solver* (with a relativistic MHD option), a few *Flux Limiters* to consistently account for radiative transfer in relativistic contexts, and other physical sources like thermal conduction (both isotropic and anisotropic), sub-resolution *turbulent stirring*, implemented also through *(semi)-implicit diffusion solvers* have been developed and integrated as additional modules.

Finally, a few *gravity solvers* have been provided: a *Particle-Particle-Mesh* (PPM) and a *hierarchical Tree solver*, the latter implementing a *parallel Barnes-Hut* spatial decomposition with a *Peano-Hilbert ordering path scheme* to enhance the parallel efficiency of the communication among different blocks within similar tree levels. These gravity solvers are particularly important in astrophysical applications because they enable a modelling of self-gravitating systems like stars, galaxies, and clusters thereof and also of the processes of stellar formation. The latter is triggered by gravitational instability arising in a finite-pressure, self-gravitating interstellar medium inside positive density fluctuations satisfying the *Bonnor-Ebert instability criterion*.

It is thus of paramount importance to implement accurate numerical algorithms to compute the gravitational interactions. This task is made difficult by the global character of gravity, a truly long-range, unshielded interaction which prevents from adopting local computational schemes.

a. Prototype modelization, structure and functional description

In figure 25. we present a flowchart of a typical **FLASH** problem, showing the code architecture.

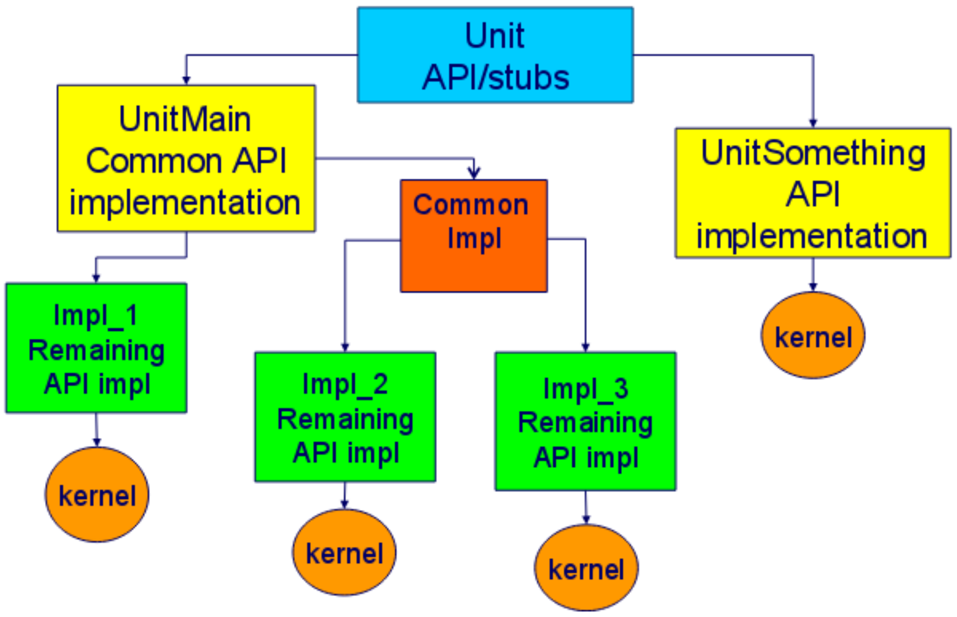


Figure 25: A schematic diagram of the FLASH units hierarchy and inheritance.

A **FLASH** unit defines its own *Application Programming Interface* (*API*), which is a collection of routines the unit exposes to other units in the code. A *unit API* is usually a mix of accessor functions and routines which modify the state of the simulation. There exists *API* for all the different required functionalities: grid decomposition, hydro solver(s), step advancement, physical input, I/O functions.
The *Main Unit* is a collector and organiser for other code units and controls the consistency of the organisation of the workflow. Some units

are common to all simulations, and thus are always present: typical examples are the *Grid* and the *I/O* units.
The modular_structure of **FLASH** guarantees that the user can build her/his own computational problem by independently assembling a set of specific modules, corresponding to specific API implementations. Note that each unit can have more than one implementation of its API. The Grid Unit, for example, has both an Adaptive Grid and a Uniform Grid implementation. Although the implementations are different, they both conform to a common standard, the *Grid API*, and therefore appear the same to the outside units. This feature allows users to easily swap various unit implementations in and out of a simulation without affecting the way other units communicate. Thus, no parts of the code have to be rewritten if the users decide to implement the uniform grid instead of the adaptive grid.
A very important feature of **FLASH** are the *stub* units. The top directory of every unit contains a stub or null implementation of each routine in the Unit's API. The stub functions essentially do nothing. They are coded with just the declarations to provide the same interface to callers as a corresponding "real" implementation. They act as function prototypes for the unit. Unlike true prototypes, however, the stub functions assign default values to the output-only arguments, while leaving the other arguments unaltered. These units provide a skeleton for the user to implement algorithms which are not carried in the default distribution of **FLASH**, like for instance initialization and boundary conditions specific to the physical problem the user wants to model.

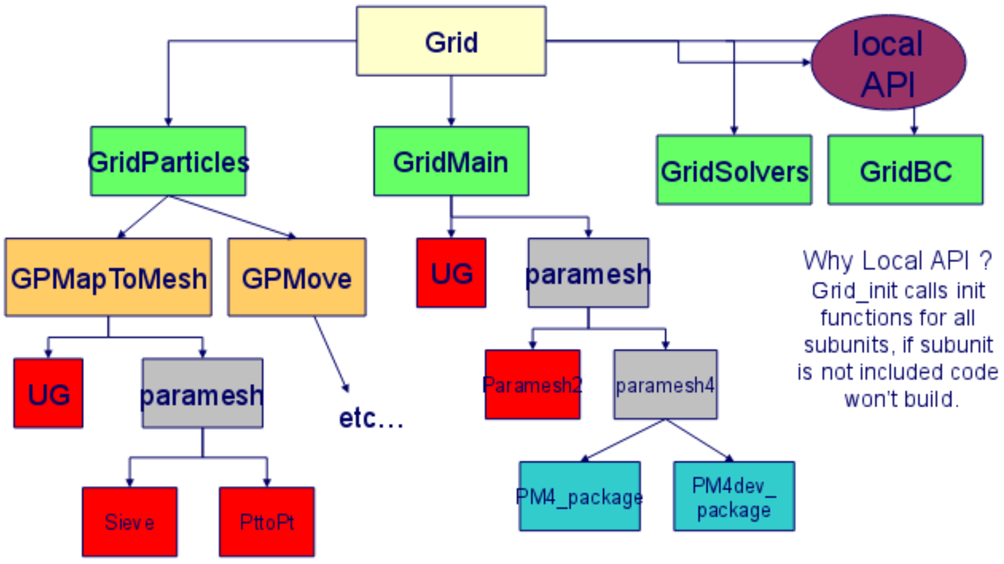


Figure 26: A simplified representation of the Grid Unit.

Figure 26 above shows a simplified representation of the *Grid* unit, which creates the spatial grid decomposition of the domain and controls its consistency. The frequent calls to the *Paramesh* routines control the consistency between (sub)grids at different refinement levels and between blocks of the same level (amon others). *Particles* here represent discrete objects like single stars or star clusters (characterised not only by a mass spectrum but also in their stellar evolutionary properties as *Simple Stellar Populations* [*SSP*]). The latter represent physically independent objects which interact with the gaseous component (the *Interstellar Medium*) both gravitationally and by exchanging energy (*stellar feedback*). The *GPMapToMesh* and *GPMove* units control these exchanges.

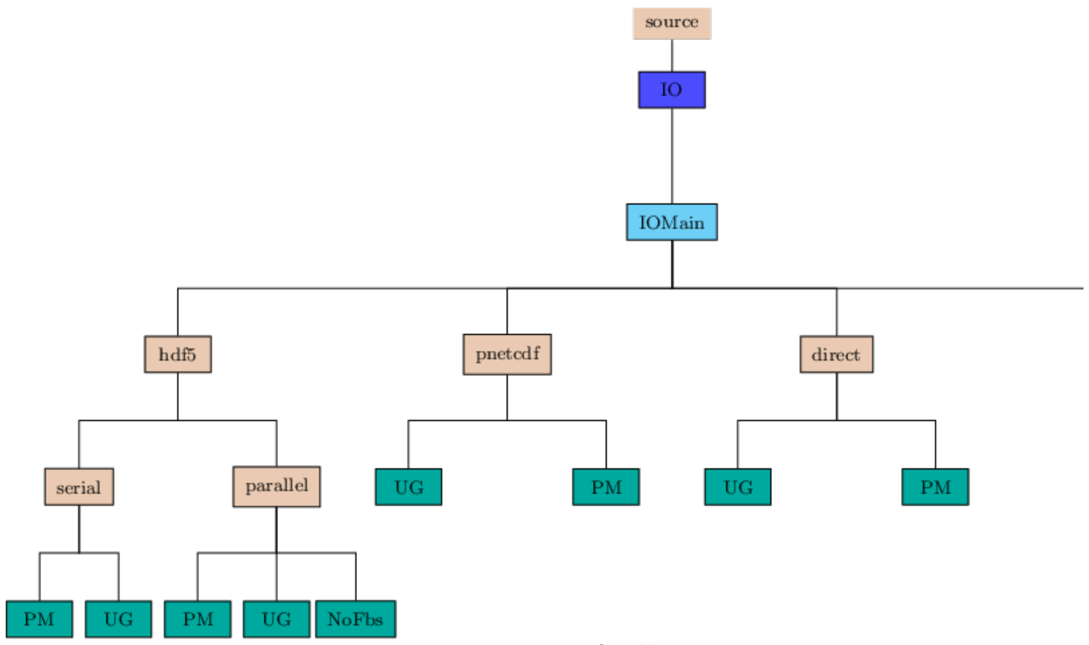


Figura 27: The I/O unit.

In this work we will focus on the *I/O Unit*, whose structure is represented in figure 27, and in particular on the *HDF5* I/O, which is one of the less optimised and critical phases [138]. **FLASH** produces two types of data files: *standard* and *checkpoint* outputs. The latter contain only a reduced set of data related to physical and grid variables needed for visualisation and analysis, while the former also contain all the intermediate data needed to restart a simulation from a given timestep. **FLASH** provides different HDF5 I/O unit implementations - the serial and parallel versions for each supported grid, Uniform Grid and PARAMESH structure. The format of the HDF5 output files produced by these various I/O implementations is identical; only the method by which they are written differs.
By default, the parallel mode of HDF5 uses an independent access pattern for writing datasets and performs I/O without aggregating the disk access for writing. Parallel HDF5 can also be run so that the writes to the file's datasets are aggregated, allowing the data from multiple processors to be written to disk in fewer operations.
Despite its modular architecture, allowing a very large flexibility in designing target-specific numerical experiments and physical simulations, some physical properties which are not directly deliverable from hydrodynamic properties are difficult to be coded in **FLASH**. One example comes from Astrophysics: the formation and evolution of stellar populations in galaxies and the derivation of their spectral properties (a typical observable quantity) represent computational tasks related but not output from the code. Instead of producing new modules in **FLASH** to undertake these tasks it turns out to be more convenient to *glue* **FLASH** together with other packages specifically designed to have stellar formation and spectral evolution capabilities, like **SYGMA** [139].
In more detail, the main reasons for this are at least two: (1) *Different spatial and temporal scales*: Star formation takes place from Jeans unstable positive density fluctuations of the ISM, whose spatial and temporal scales are orders of magnitudes smaller than those related to the *global* evolution of the galaxies where they are contained; (2) *Code recycling*: There exists already free software packages which compute the formation of stars from the ISM and the spectral evolution of *Simple*

Stellar Populations (*SSPs*), the building blocks used to model the stellar properties of galaxies.
The first characteristic makes it highly inconvenient to build *ad-hoc* modules for stellar evolution within **FLASH**: the large timestep differences between CFD and stellar formation timescales will result in a significant workload imbalance. An alternative would be to have a parallel run where at each FLASH timestep star-forming regions are identified and their data are sent as input to packages like **SYGMA** which will compute the properties of (newborn and already present and evolving) SSPs, giving back to **FLASH** the radiative and thermal outputs which represent the *stellar feedback* on the evolution of the galaxy's *ISM*.
We are currently developing a workflow where **FLASH** and **SYGMA** will run concurrently and asynchronously and perform different tasks: the latter will evolve the stellar components and calculate their spectral evolution, while **FLASH** will continue to be perform the CFD calculations related to the non-stellar components (including Dark Matter). The two components will synchronise periodically their outputs for physical consistency. This will allow a *more efficient* simulation strategy, where independently optimised parallel codes will run independently but stay loosely coupled, thus mimicking the actual *loose* physical coupling between stellar components and the ISM.

b. Actual implementation

FLASH is mostly coded in F90, with some parts coded in C, C++ or Python for higher efficiency. A *setup* Python script is used to arrange the required units into a *compilation directory* (user specified) where links to the actual units and routines needed to produce an executable are collected.
Parallelization is implemented either with MPI and/or OpenMP. Specific paths to the actual libraries (including those specific to I/O) are stored in a site-specific configuration file, where compilation and link-specific options can be set. **SYGMA** is coded in Python and it has recently been evolved to v. 3.x.
We plan to investigate fast I/O techniques made available within the present collaboration to improve the I/O both of the large datasets produced by **FLASH** and during data exchange between **FLASH** and **SYGMA**. More specifically we will use:

- ***CAPIO (UNIPI+UNITO)*****:** We would explore the effectiveness of integrating within the **FLASH-SYGMA** workflows the CAPIO middleware to boost its I/O performances without modifying the original codes (see D4.FL3).
- ***Nethuns (UNIPI)***. Additionally, we will investigate the feasibility to integrate the lightweight userspace library Nethuns that offers a straightforward programming model for network I/O and test the available I/O accelerations frameworks, nicknamed engines, as a backend (see D4.FL3).

c. Validation tests and results

The validation phase has just started. We plan to perform two tests:

- Recompile HDF5 libraries used by **FLASH** with the ***Nethuns*** primitive calls, and compare the efficiency for different computing platforms, including generic clouds.

- Integrate ***CAPIO*** in the **FLASH**-**SYGMA** workflow and perform a series of tests with computational problems of increasing complexity.

2.7.4 Prototype evolution and implementation

a. Prototype evolution direction

The prototype described above is at TRL2 (experimental proof of concept). The further steps of evolution will aim at reaching two targets:

- Implement effective I/O capabilities within the workflow and testing over a significant range of different platforms, from Linux clusters to more efficient and homogeneous HPC systems up to a cloud environment, to evaluate the effectiveness with increasing complexity in terms of weak and strong scalability.
- Implement at least one similar workflow where **FLASH** is loosely coupled to packages different from **SYGMA**, related to problems arising e.g. in plasma confinement physics.

b. Prototype Implementation and involved tools

As described above, we will exploit two main tools made available within the current collaboration: **CAPIO *(*UNIPI*)*** and **Nethuns *(*UNIPI*)***, see D4.FL3 in sections, respectively, 2.4.3 and 2.4.2.
CAPIO will be used to allow communications between two loosely coupled codes, i.e. **FLASH** and **SYGMA**. The coupling between them arises from the need to update a subset of data from the former, related to spatial regions within a galaxy whose physical state is heavily affected by a component (stars) whose properties are concurrently computed by **SYGMA**. We plan to use *CAPIO* to avoid checkpointing from inside **FLASH**, which would require developing new modules. We foresee a higher versatility allowed by this methodology, particularly when applied to highly inhomogeneous cloud computing environments.
Nethuns will be used to improve the I/O of the large data outputs produced by **FLASH**, i.e. of both checkpoint and data files. This is particularly relevant as parallel I/O is seen as one of the major bottlenecks in the exploitation of **FLASH** capabilities on future exascale architectures.

2.7.5 Final validation tests

The final phase of this validation will consist in a systematic exploration of the scaling properties of two different products:

- The **FLASH-SYGMA** workflow exploiting the *CAPIO* library to loosely communicate data related to the stellar component in global galaxy formation and evolution numerical experiments.
- The I/O of **FLASH** of large datasets with the HDF5 library compiled using *Nethuns* to deal with low-level network communications.

This phase will bring the prototype from the current TRL2 to a higher level and will open the door to the exploitation of the CAPIO and Nethuns libraries in state-of-the-art, scalable applications (both scientific and technological) on exascale architectures.

2.8 WorldDynamics.jl

2.8.1 Introduction

WorldDynamics.jl is a Julia framework for world dynamics modeling and simulation. It is an open-source Julia package which aims to provide a modern framework to investigate Integrated Assessment Models (IAMs) of sustainable development benefiting from Julia's ecosystem for scientific computing. Its goal is to allow users to easily use and adapt different IAMs, from World3 to recent proposals.
An IAM aims to integrate the key aspects of society and economy with the biosphere and atmosphere within a unified modeling framework. The main goal is to provide informed policymaking in different contexts such as climate change, human development, and social development. Each model spans multiple disciplines including, but not limited to, economics, energy systems, agriculture, technology, etc. Generally speaking, each model can be seen as a set of subsystems.
Integrated assessment modeling allows us to estimate what possible future scenarios look like and to evaluate possible policies. To quantify the outcomes, numerical models are employed.
Different models were proposed in the last fifty years although they are not suitable to capture the changing nature and complexity of today’s economic realities due to climate change.
Despite the current (and future) situation, understanding how each model works as well as their outcomes, is one of the major open problems. Even though almost every proposed model is freely available, actual implementations leverage proprietary software.
Through the means of Julia, we have provided a modern framework to investigate models of global dynamics focused on sustainable development based on current software engineering and scientific machine learning techniques.
In particular, our group is developing a Julia library to allow scientists to easily use and adapt different world models, not only the already cited WorldX but also recent proposals.

2.8.2 Related works

One of the most difficult and pressing questions that science has faced is trying to predict the evolution of human society in terms of its basic aspects, such as capital investment, food production, natural resources, population size and pollution. These efforts have been methodologically revolutionized by the use of computers in modern times. A historic step in this regard was the development of the World3 model [142], considered to be one of the most influential computer simulations of socio-economic systems [143]. To date, several models have been proposed and have largely influenced the scientific debate around crucial questions on policy making.
However, even the most recent models have been developed using software that is not freely and widely available, such as the DYNAMO language dating back to the late 1950s and the proprietary software Stella and Vensim. Moreover, they rely on methods that do not exploit modern approaches to scientific computing in general [141]. Some independent implementations in different languages have been provided by researchers [144], but they only deal with certain parts of these models.

World2 and World3, two well-known system dynamics models, have been implemented in several programming languages and simulation environments, beyond the popular Vensim and Stella. For instance, Simulink and Modelica have been used to implement these models. There are also several implementations of these models in different programming languages. In Python, several repositories have code implementing some of the IAMs proposed by the Club of Rome. The most comprehensive description and implementation of the World2 and World3 models in Python can be found in [149]. [147] describes an implementation of World2 in R, while [146] provides an implementation of the same model in C++. The Julia ecosystem also offers some IAMs implemented using the Mimi framework, such as MimiPAGE2009.jl, which implements a model for estimating the social cost of carbon emissions [148]. ClimateMARGO.jl is another Julia package that implements an idealized framework for optimizing climate control strategies by implementing the MARGO model [145].

2.8.3 Actual prototype description and maturity level

The tutorial included with WorldDynamics.jl serves as an introduction to the package's key capabilities. In short, it highlights the following features:

- The ability to recreate all the figures found in books that detail the World1, World2, and World3 models.
- The option to perform sensitivity analysis by adjusting the initial values of variables.
- The capacity to analyze alternative scenarios by modifying either the model's parameters or the interpolation tables, which are utilized to approximate non-linear functions through linear segments.

a. Prototype modelization, structure and functional description

We have already implemented some preliminary Integrated Assessment Models (IAMs) such as the model by Forrester's World2 and Meadows et al. 's World3.
The models are available in WorldDynamics.jl. In particular, we provide a modular implementation where each model is composed of several subsystems implemented as a Julia function and are later composed to provide a full system. Each subsystem can be analyzed independently and can be modified to express different states.
As of today, we can reproduce several figures of the book Dynamics of Growth in a Finite World and more generally the same outcomes. Moreover, we can adapt the model to more modern data. Thus, we can also validate a proposed model studying the outcome of the model and comparing it with the ongoing development of the real world.
Our project also allows us to perform sensitivity tests by simply modifying the parameters or the interpolation tables without touching the underlying models. We can also substitute a description of a subsystem (i.e., a set of equations) with a different description.
WorldDynamics.jl leverages several Julia’s packages such as DifferentialEquations.jl and ModelingToolkit.jl which composes differential-algebraic systems of equations and thus, the structure of an IAM whose variable interactions are modeled by differential-algebraic equations.

The large size of these systems and subsystems results in a set of equations that is also of considerable magnitude even for simple models. To speed-up the resolution phase parallel solver must be used.

b. Actual implementation

The project, which is available at [153], is implemented mainly in Julia, adopting a modular approach. It includes the implementation of the most famous models such as the entire WorldX series of models of the Club of Rome.

The current version of the code allows the user to extend the proposed model by changing each subsystem independently of any other subsystem of the same model.
New model can be implemented by providing the required equations and coefficients. In particular, WorldDynamics.jl requires the set of parameters and interpolation tables as well as the initialization data. Each of the previous is a .jl file. Then the set of equations gives a subsystem; all the subsystems are composed to implement the actual model.
By changing the data (parameters, interpolation tables or initialization data) it is possible to make predictions and analysis of different policies given the model.
The plot operation allows a visual representation of each subsystem throughout the time giving an interactive view to the user. The plots produced include a curve for each equation of the system that together fully describe the model given the initial data and parameters.
WorldDynamics.jl allows a quantitative analysis of each model and sensitivity test by changing the parameters, interpolation tables of the variables in the specific. jl file then, creating the new system and solving it.

c. Validation tests and results

WorldDynamics.jl was tested against the published models and the available implementations [152]. The main goal of the test was to show the correctness of the implementations by recreating the same results already published of the well-known models.

2.8.4 Prototype evolution and implementationCompleto

The project roadmap includes the implementation of more historically relevant MEIs, such as Nobel Laureate William Nordhaus' DICE model and the recent Earth4All model. We are currently working on the latter model which is already implemented but not modularized before including a parallel implementation of the solver as well as a first automated model generation through DataDrivenDiffEq.jl [151].

a. Prototype evolution direction

Julia employed several libraries which offer the possibility to leverage parallel (GPU) solvers out of the box. A crucial aspect requires the usage of DataDrivenDiffEq.jl; the library allows us to cope with finding the best fitting dynamic systems through machine learning algorithms which in turn, have to be parallelized as well.

Then, WorldDynamics.jl will be extended to exploit parallel implementations and data driven dynamical models' computation.
Specifically, we will start using Parallel Ensemble Simulations supplied by DifferentialEquations.jl to make use of GPUs and speed-up the resolution of the system of equations.
On the other hand, the World Bank [150] provides a plethora of datasets for which we aim to develop forecasting models exploiting machine learning algorithms to generate suitable coefficients and equations. The size of the said dataset requires careful implementation and HPC tools. We will use what The World Bank provides to generate new models through machine learning. In particular, we will use regression algorithms to learn functions (instead of coefficients as in linear regression). This is a novel approach to model making: until now, every model was developed 'by hand', a non-scalable approach that leads to simple models. The extension will transform WorldDynamics.jl in a tool to generate models, compare them with themselves and the data. Moreover, WorldDynamics.jl will provide a flexible framework that allows the usage of several solvers and integration with different methods with a reduced effort. WorldDynamics.jl will allow model construction in a simplified way while enabling the application of modern scientific computing techniques over new and classical models as well as employing machine learning techniques for model design.

b. Prototype Implementation and involved tools

The library is developed in the Julia programming language, making use of the ModelingToolkit.jl and DifferentialEquations.jl libraries.
Moreover, DataDrivenDiffEq.jl [151], a library for finding systems of equations automatically from a dataset, will be used as a first tool to exploit WorldBank's data to generate a set of functions that fits the input data. The main goal is the automatisation of model development.
These tools include automatically discovering equations from data and using this to simulate perturbed dynamics.
We would like to investigate the workflow notion to develop a more accessible framework. WorldDynamics,jl would benefit from it for giving a more readable access to the models as well as its workflow. This implies a possible cooperation with:

- **Jupyter Workflow (UNITO)**: Jupyter born as a native Julia notebook. We want to investigate how the tool can be adapted to make WolrdDynamics.jl a more accessible framework and to, possibly, execute our model in a distributed fashion. Here the goal is to improve the performance and readability while exploiting our own HPC architecture (supplied by the involved entity INRIA), see D4.FL3 section 2.1.5.
- **BDMaaS+ (UNIFE)**: our tool provides the means to execute several simulations of the same model with different parameters together with runs of the model with different subsystems. We aim to exploit BDMaaS+ framework to speed-up the process and run different models (and simulations) in a parallel fashion. The nature of our tool allows the exploit of BDMaaS+'s properties for computing the best configurations. Moreover, we can further exploit the available infrastructure and cloud computing services (i.e., Amazon EC2) for an hybrid approach (See D4.FL3 section 2.1.1).

For our future extension, we can benefit of a possible partition of both simulation and model discovery based on data utilization.

In addition, we would like to investigate Machine Learning techniques and the data furnished by real-time simulators. Which means:

- **aMLLibrary (POLIMI)**: we want to take inspiration from their techniques for our machine learning tasks. Furthermore, we would like to investigate their tool as a possible validation for our models since the tool is an independent implementation. Indeed, our main task is model discovery whose base case is regression for which they already gave an implementation. We also think their tool can make a better exploitation of the available data. See D4.FL3 section 2.1.8.
- **Real-Time Simulator for Digital Twin and Hardware-In-Loop in the Electrical Power Networks Scenario (UNIBO):** we would like to investigate their model. Especially the output their simulation provides to plug-in in our model as a source of new data and hence equations. In particular, the models until now proposed are generally based on coarse-grain data. We may use their tool to build a more precise subsystem alongside to extrapolate useful data for global consumption. See D4.FL3 section 2.1.3.

2.8.5 Final validation tests

Unlike what has been done until now, we will shift the focus on the evaluation metrics: the *KPIs* used mainly rely on performances such as response time of the simulations and model generations. We will continue to validate the correctness of our implementations especially for the data-drive model discovery. In that case, tests against the already implemented model will be performed.

2.9 Optimized deployment of cloud-native applications over multi-cloud and cloud continuum scenarios

2.9.1 Introduction

Orchestration solutions such as Kubernetes are becoming useful tools for HPC users as they seek to deploy and manage increasingly complex workflows across a wide range of computing resources, including multi-cloud and cloud continuum scenarios. Multi-cloud and cloud continuum scenarios refer to a plethora of interconnected computing resources consisting of cloud, edge, and on-premises resources, usually located in different locations with possibly different ownership, renting prices, and sizes.
However, the high heterogeneity of the cloud-continuum introduces many challenges from the service management perspective, such as identifying optimized deployments for HPC applications that might require high computational resources. To select an optimized deployment, there is the need to explore multiple configurations and to evaluate their performance. On the one hand, a service provider would like to deploy its application by analyzing the pricing perspective, thus looking to rent those resources that can minimize the overall provisioning costs, i.e., the renting prices of the chosen execution environment, such as Kubernetes clusters or vanilla Virtual Machines (VMs). On the other hand, communication latencies

between different computing locations might play a crucial role in assessing the performance of complex workflows, such as the ones of HPC applications.
Furthermore, the actual deployment of these applications upon the cloud continuum is another challenging task. Connecting resources in a multi-cloud scenario requires creating an overlay network capable of interconnecting computing resources located at different cloud facilities in a transparent fashion for both application providers and end-users. Finally, state-of-the-art orchestration components are not designed to take into account all these requirements and to fully benefit from the capabilities of multi-cloud and cloud continuum scenarios.
Therefore, it is very difficult to select an optimized configuration of computational and network resources of complex HPC applications before their actual deployment. There is the need to develop novel solutions capable of continuously exploring different configurations from the ones available, and to shift from one to another according to application-specific requirements, and the current resource availability. To fulfill this task, there is the need for novel solutions capable of operating at many levels, from the simulation of complex cloud-native applications in a cloud-continuum scenario to the actual deployment and orchestration of these applications.

2.9.2 Related works

Services and resource management in the Compute Continuum is a challenging research topic that calls for innovative solutions capable of managing the multiple layers of computing resources. Even if several efforts have been made in the single cloud scenario, multi-cloud and cloud continuum scenarios are still unexplored. Related efforts focused on the requirements of this project, without trying to address them in a comprehensive manner. Instead, with the development of this prototype we aim to address the optimized placement of cloud-native applications at many levels: federating multiple clouds, running what-if scenarios analysis to find the most appropriate allocation of resources, and orchestrating services.
Among these works, [154] proposes a resource orchestration framework called ROMA to manage micro-service based applications in a multi-tier computing and network environment that can save network and computing resources when compared to static deployment approaches. In [155], Pereira et al. propose a hierarchical and analytical model to overwhelm the resource availability problem in Cloud Continuum scenarios. They present multiple use cases to demonstrate how their model can improve the availability and scalability in edge-cloud environments. Moreover, the authors in [156] describe a model-based approach to automatically assigning multiple software deployment plans to hundreds of edge gateways and connected Internet of Things (IoT) devices in a continuously changing cyber-physical context.
On the other hand, Digital Twin (DT) approaches are gaining momentum as they became a could represent useful tools to enable what-if scenario analysis [157] of applications running in multi-cloud scenarios. This will implement a faster (and parallelizable) process for the exploration of a larger number of configurations, and even allowing the rapid prototyping of custom Kubernetes functions (e.g., autoscaling, scheduling). As a result, DTs could be very effective in speeding up the parameter identification

process, as well as in significantly broadening its scope, with potentially significant costs savings [158, 159].

2.9.3 Actual prototype description and maturity level

At the current stage, we are working at the design level (TRL 2). The proposed prototype is an integration of three different tools: BDMaaS+, INDIGO, and Liqo. This integration aims at enabling an optimized deployment of complex cloud-native applications over multi-cloud and cloud continuum scenarios by exploiting the capabilities of multiple and distributed computing clusters.
While the prototype is still in its design stage (TRL 2), the tools involved in this project have different maturity levels. Specifically, we can summarize the TRL of each tool as following:

- **BDMaaS+** was developed as part of many research projects; we validated it experimentally in articles published in international journals and conferences. Therefore, we can claim that BDMaaS+ has a TRL of 3. See D4.FL3
- **INDIGO** has a TRL of 5, as it was experimentally validated in laboratory settings and other use-cases. See D4.FL3
- **Liqo** has a TRL of 5, as it was experimentally validated in laboratory settings and other use-cases. Moreover, Liqo is in production at the Politecnico di Torino, to scale the computational resources when needed, for example, in all those situations that require an overbooking of VMs e.g., during the final examinations of the university’s courses. See D4.FL3

a. Prototype modelization, structure and functional description

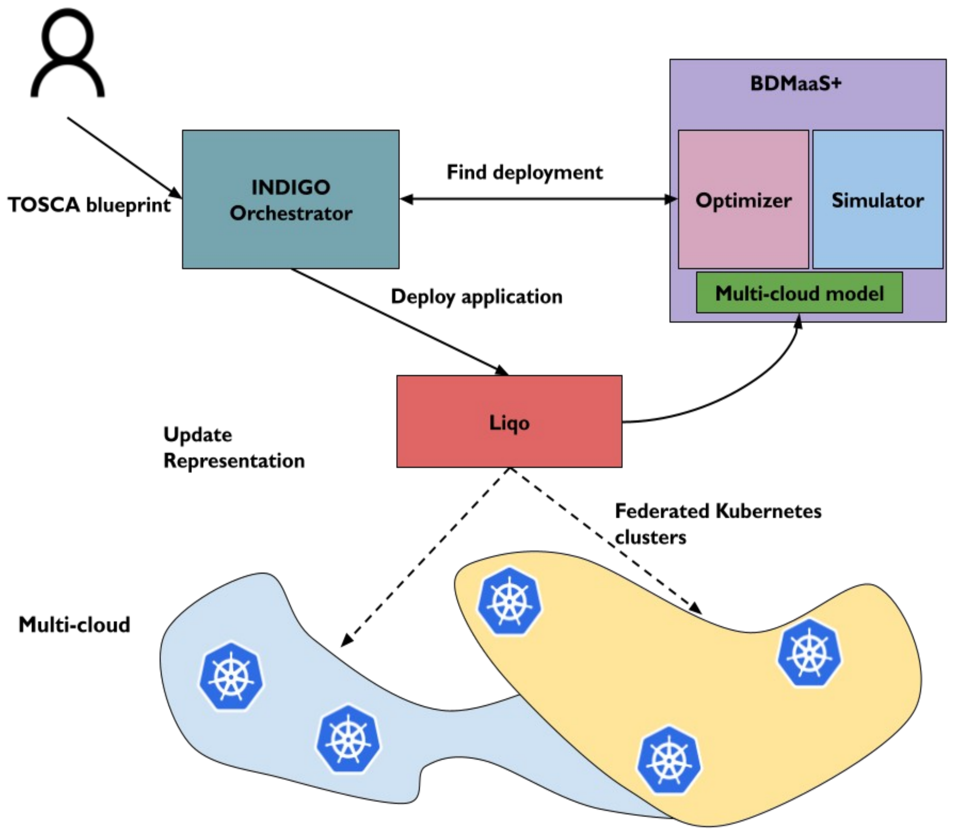


Figure 28: Applications deployment in a multi-cloud scenario using INDIGO, BDMaaS+, and Liqo.

To solve these challenges, we present a novel approach based on several contributions as illustrated in figure 28. Specifically, figure 28 shows the use-case of an application provider interested in deploying a HPC application. To do so, the application provider needs to describe the application and its workflow using the standardized TOSCA notation. Then, INDIGO orchestrator interacts with BDMaaS+ to find the most appropriate set of computing resources considering the application requirements, application provider defined policies (pricing, latency), and the current availability of resources among the multi-cloud.

To do so, BDMaaS+ implements Digital Twin methodologies to enable an accurate representation of applications operating in multi-cloud and cloud continuum scenarios. For creating this virtual representation, BDMaaS+ makes use of input static description of an application (TOSCA blueprint) and the state of resources available across the multi-cloud. By capturing the state of an existing HPC application through a virtual representation of the HPC application it would be possible to run simulation-based accelerated timescale analysis and to select a proper deployment description.

Then, BDMaaS+ returns to INDIGO the information on the actual computing resources that can sustain the QoS demanded by the composite cloud application described in the TOSCA blueprint. With such fresh data, the INDIGO orchestrator will produce an application deployment plan that includes a set of Kubernetes “intents”. The orchestrator will then enforce the application provisioning (i.e., the deployment of all software modules the application is composed of) by invoking the Liqo API and providing it with the above defined Kubernetes intents.

Guided by the deployment requests issued by the INDIGO orchestrator, Liqo will dynamically create a federation of networked computing resources, then it will take care of instantiating, configuring and running the application's distributed components in the federation. The advantage of Liqo compared to alternative solutions is the capability to create a unique virtual cluster spanning across multiple physical infrastructures, hence simplifying the deployment and the management of applications within, which behave the same way independently from their actual location.

b. Actual implementation

Considering that BDMaaS+ makes use of a simulation-based approach, to shorten the time required for what-if scenario analysis it could be useful to run parallel simulations (e.g. a simulation per thread). We plan to develop a parallelized implementation of the BDMaaS+'s Optimizer component for exploiting the parallelism of multiple CPUs and GPUs when available. This will speed up the optimization process, thus allowing BDMaaS+ to promptly find a new configuration in a shorter time. To do so, we envision to adopt computational intelligence approaches running parallelized simulations, each one considering a different configuration. Furthermore, we envision that other kinds of methodologies such as bayesian optimization could be used as optimization black-box.

c. Validation tests and results

To validate the prototype, we plan to test the deployment of several applications and to collect their metrics using Liqo. Then, we will compare these results with the ones obtained through vanilla orchestration solutions that do not take into account the requirements of multi-cloud and cloud continuum scenarios. We expect that the prototype will overperform vanilla orchestrators from different perspectives: provisioning costs (USD per day), expected latency, and overall application performance.

2.9.4 Prototype evolution and implementation

a. Prototype evolution direction

During the project, we will implement the described prototype. Specifically, we are planning to implement all those functions to support the interoperability between BDMaaS+, INDIGO, and Liqo. This part of the project will require many efforts both from the design and the implementation levels. Furthermore, we need to implement all those functions that can be triggered by APIs calls. Among these, extending Liqo's capabilities of collecting monitoring data from a multi-cloud / cloud-continuum setup will be essential to allow BDMaaS+ to create its accurate Digital Twin mode.

b. Prototype evolution structure and description

The prototype evolution process will follow an iterative and incremental approach, with each iteration building upon the previous version to gradually refine and enhance the prototype. This is to create a more mature and robust design capable of fulfilling the desired requirements. We believe that at the end of the prototype evolution process, our prototype will have a TRL of 5.

c. Prototype Implementation and involved tools

As illustrated in figure 28, this prototype will include three different tools from different partners: **BDMaaS+** from the University of Ferrara (see DF4.FL3, section 2.1.1) the **INDIGO** orchestrator from the University of Bologna (see DF4.FL3, section 2.3.2) and **Liqo** from the Politecnico di Torino (see DF4.FL3, section 2.3.3) . With this prototype we aim to create an a comprehensive solution that leverages multiple technologies to enable efficient orchestration in multi-cloud and cloud continuum scenarios.
With regard to the implementation, we are currently working to extend the BDMaaS+ framework to support the simulation of cloud-native applications over multi-cloud and cloud continuum scenarios. This will enable an accurate modeling of complex and multi-workflow applications and their deployment on a plethora of computing resources distributed among multiple locations. BDMaaS+ will implement an optimization framework capable of finding a suitable location in terms of pricing, latency, and other user-defined Key Performance Indicator (KPI). Finally, another extension will entail the support for continuous optimization by leveraging the monitoring information provided by Liqo.
To support the proposed integration, we will extend the INDIGO orchestrator to interact with the BDMaaS+ framework in the aim of requesting and obtaining a list of computing resources that can support the SLA demanded by the requesting user. Also, the tool will be enhanced with new capabilities to combine the information found in the TOSCA blueprint of the service with the one received from the BDMaaS+, leverage this data to bake deployment plans in the form of Kubernetes intents and feed them to the Liqo platform. At the same time, we will extend Liqo to support the discussed use-case. Specifically, Liqo will extend its northbound API to better support the INDIGO orchestrator and BDMaaS+, and it will be extended to gather new monitoring metrics at run-time (network, compute, storage) in order to feed BDMaaS with more accurate information for its prediction.

2.9.5 Final validation tests

As final validation tests, we will evaluate our prototype in a large-scale multi-cloud environment to verify the feasibility of the approach on real-world case studies. As part of this validation, we would like to show how the desired prototype can implement continuous re-allocation of computing resources in many computing clusters and to adapt to the current workload or environmental conditions.

2.10 FastFlow: an alternative programming model for HPC applications

2.10.1 Introduction

FastFlow [160] is a structured parallel programming framework that supports application programmers in the process of building efficient

parallel applications by creating composition of parallel patterns specialized through proper business logic code parameters and system programmers in the process of building new specialized, possibly domain-specific parallel patterns through the composition of parallel building blocks.
The entire framework is provided as a header only library that must be compiled with the user provided business logic code to obtain the executable. It is being developed since early '2000s and maintained by the University of Pisa and the University of Torino, in Italy, and provides both stream and data parallel patterns, usually composed into parallel applications according to the two-tier rule introduced by Kuchen in [161]. Initially, FastFlow has been designed to target shared memory multicore architectures only. Later, the possibility to offload and orchestrate computations to different kinds of accelerators has been introduced, as well, most recently, to target classic cluster architectures such as the ones typically in the the top500 and green500 lists.

2.10.2 Related works

Several parallel programming frameworks have been proposed, based on the concept of structured parallel programming, either according to the algorithmic skeleton viewpoint [173] or according to a more software engineering viewpoint such as the one centered on design patterns [174]. Among all those proposed, the ones still being maintained and used, also in the framework of different research projects we mention Muesli [169] that provides support to target both shared memory and cluster architectures, SkeTo [170] that introduced automatic optimizations of data parallel computations through conscious exploitation of a map fusion refactoring rule, OSL [171] that brought into structured parallel programming the BSP model, and finally SkePU [172] that also efficiently supports GP-GPU accelerators. To the best of our knowledge, Muesli and SkePU are the ones still used although not explicitly being mainstream in HPC.

It is worth pointing out that several widely used libraries and frameworks include concepts from the structured parallel programming area. IntelTBB provides some parallel patterns similar to the ones provided in FastFlow and other structured parallel programming frameworks, although the computation patterns are provided at the very some level of mechanisms that can be used to program other parallelism exploitation configurations [175]. Microsoft included a lot of patterns in is own .net parallelism library [176].

2.10.3 Actual prototype description and maturity level

The key idea backing up FastFlow is that programmers (application developers and/or system programmers) may only express parallelism through the usage of available parallel building blocks. Each building block encapsulates all what's needed to exploit a given well known, reusable, composable and parametric parallel pattern that must be instantiated and specialized through the provision of the business logic code implementing the user/programmer specific logic [162]. As an example, a data parallel computation may be expressed either using a "parallel for" pattern, similar to the classical OpenMP pattern, or a "map" pattern, expressing the very same kind of parallel computation in a way the data parallel computation may be used as parameter of other patterns (e.g., as a pipeline stage or as a farm worker). In both cases, the computation to be implemented in the single iteration (parallel for) or onto all the items of the input collection (map) is to be provided as a function business logic parameter. Figure 29 outlines the typical workflow relative to the development of a FastFlow parallel application.

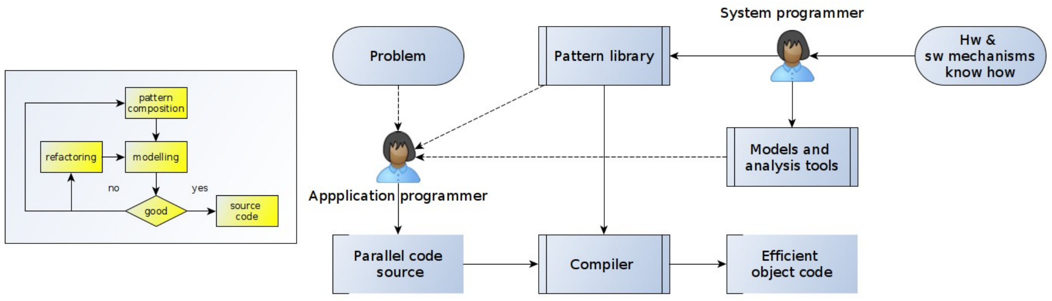


Figure 29: FastFlow application design

As such, FastFlow does not allow to express arbitrary DAGs of parallel computations with dependencies, but only those DAGs deriving from the composition of the DAGs of the different patterns used in the composition expressing the parallel application.
Several optimizations are implemented in FastFlow, that make the efficiency and performance achieved in the execution of parallel applications comparable or even better than the efficiency achieved when using more classical parallel programming frameworks.
In [163] FastFlow has been used to re-implement the ParSec benchmark suite and two distinct results are shown:

- Expressive power provided by the availability of parallel patterns is much better than the one provided by other classical parallel programming environments (less lines of code, especially in the case of complex parallel patterns, suitable to be modelled by compositions of primitive parallel forms).
- Performances achieved are close and, in some cases, better than those achieved using the other benchmarks implementations (e.g., using TBB, OpenMP or plain Posix threads, see figure 30).

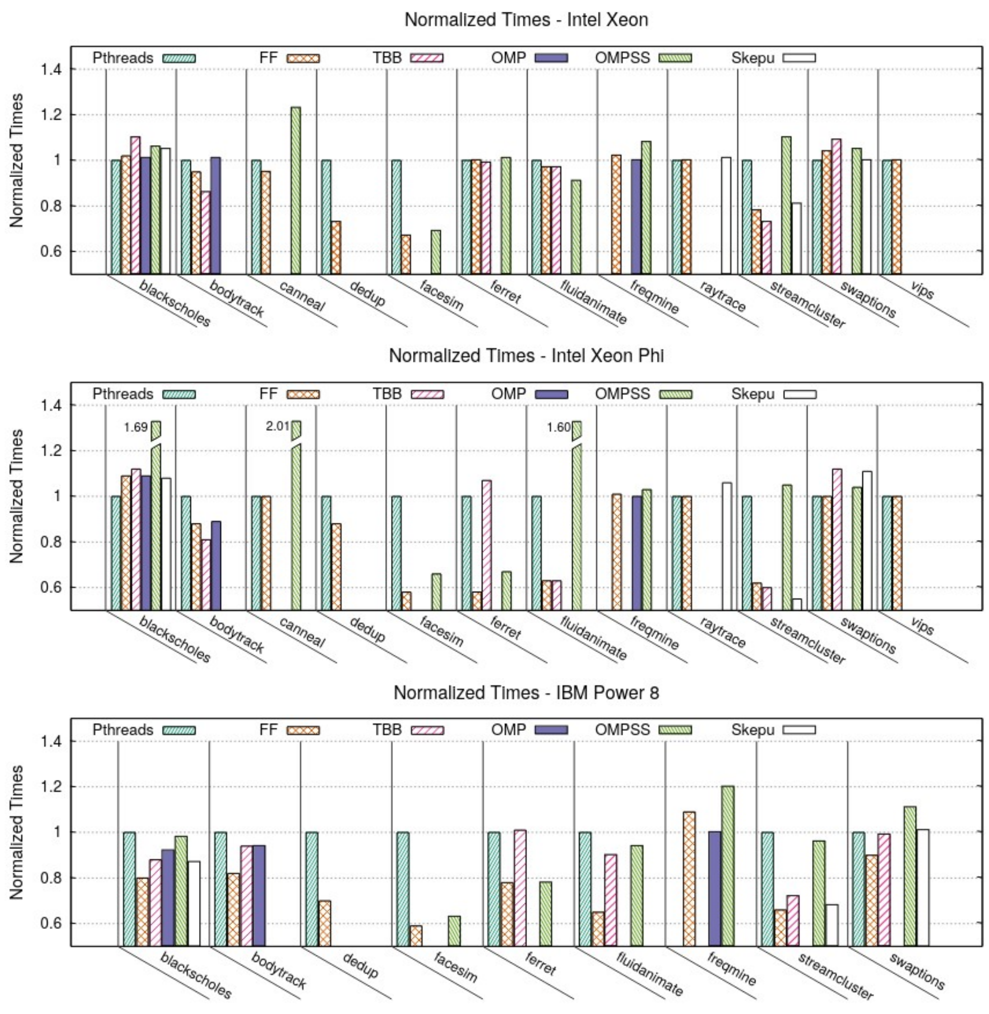


Figure 30: Best execution times normalized with respect to the PARSEC reference (from [163])

Accelerators have been included since the very beginning in the FastFlow environment.

First of all, FastFlow provides ways to use (compositions of) parallel patterns as an accelerator on a single shared memory, multicore architecture, in all those cases the application does not use all the computing engines (cores) available. In this case, computations can be offloaded to a pattern (or a composition of patterns) through a classical mechanism providing calls to seamlessly send the input data items to the accelerator and retrieve the results without any other intervention from the programmer, but the functional expression of the parallel accelerator expressed through a parallel pattern (composition).

FastFlow also manages to offload tasks to classical accelerators. GPUs are targeted with different, specialized versions of patterns, targeting GP-GPUs either using OpenCL or CUDA. The specialized patterns provided look like classical FastFlow data parallel patterns, but in addition the application programmer must somehow provide a vector of addresses and lengths of all the data items (input and output) needed to run the GP-GPU kernel(s) through offloading. Although not being completely transparent, the FastFlow GP-GPU offloading code is definitely higher level than the code usually needed to program and execute kernels on the GP-GPUs.

As far as FPGAs are concerned, FastFlow has been extended in two different ways during two different EU funded research projects. In REPARA (late '2010s), a mechanism like the one used to offload tasks to the GPUs has been used to offload computations to kernels (different kernels and possibly with multiple instances) implemented on the FPGA through classic FPGA programming frameworks (Xilinx Vivado, at that time). More recently, in the EU HPC TextaROSSA project, FastFlow has been extended with a particular "sequential" pattern (that can be used as component of other parallel patterns) to seamlessly offload computation to pre-compiled FPGA kernels on the available FPGA boards [165]. The FPGA kernels are compiled using XILINX/AMD Vitis HLS toolchain, in this case. Figure 31 outlines how the FPGA offloading node has been implemented in FastFlow.

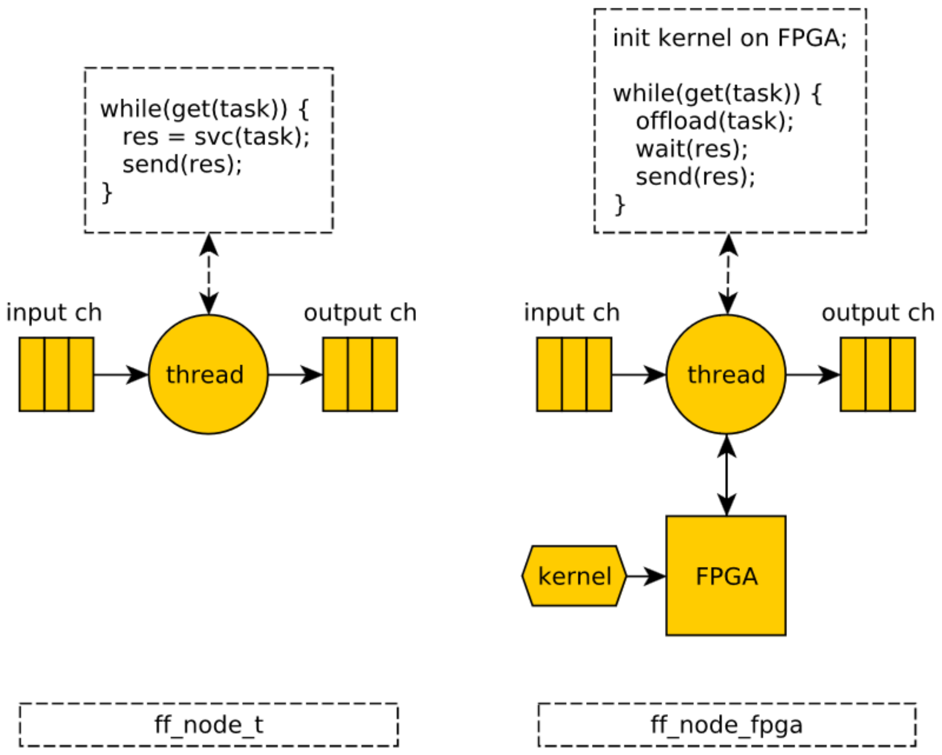


Figure 31: FastFlow FPGA offloading node

Finally, FastFlow has been recently extended to target clusters [164]. The back end has been rewritten and extended in such a way that the components of the application pattern composition may be executed of different nodes of a cluster (or workstation network). Two different backends have been implemented, one using plain TCP/IP and the other one using MPI to implement distributed synchronizations and communications. The application programmer is only asked to add a few lines of code in an application already running on a single shared memory machine, adding components to different "groups" and then to provide, outside the application, a JSON file with hints on the distribution of the groups onto the available cluster nodes (this second part is going to be made transparent to user, indeed).
The possibility to write programs using the MPI subsystem to implement intra node communications and synchronizations *de facto* implements yet another MPI+X programming model where indeed the "MPI" part is only managed by FastFlow and both the distributed and shared memory parts

of the application code are indeed programmed uniformly using the FastFlow patterns and parallel programming abstractions.

2.10.4 Prototype evolution and implementation

Within the FL3 activities distinct activities are planned with FastFlow:

- Consolidation of the distributed (COW/NOW) support
- Cloud/HPC integration through FaaS offloading
- HPC continuum offloading support
- Support to the parallelisation of flagship demonstration that will be outlined in the following sections.

Consolidation of the distributed (COW/NOW) support

The currently available distributed implementation of FastFlow targeting COW/NOW will be further refined and engineered. In particular, we aim at refining methods and tools to automatically support the distribution of groups of components of the FastFlow application onto the available cluster nodes. This activity will mainly be implemented at UNIPI, with the support of UNITO.

Cloud/HPC integration through FaaS offloading

In [166] we already experimented with the possibility to offload computations from a shared memory multicore to cloud from within a FastFlow application. Within FL3 activities, we will investigate the possibility to use the FaaS framework described in section 2.6 as offloading target, and in particular to assess:

- The possibilities offered by the elastic management of FaaS infrastructure in those cases where FastFlow faces the problem to tackle phases with substantially different computation power requirements which must be alternatively dealt with variation of the number of instances of the FastFlow pattern components that may be not easy to implement efficiently.
- The possibilities offered by the orchestrated coordination of HPC and cloud resources, in the aim of the original goals of this Flagship 3. This means that, even in case of non-varying computation requirements, cloud and cluster resources may be orchestrated to achieve the overall computation performance and efficiency goals. This opportunity will also be investigated taking into account power consumption, in addition to performance.

This task will be implemented by UNIPI interacting with ROMATOV affiliates.

HPC continuum offloading support

The possibilities offered by FastFlow to offload computations to different kind of accelerators will be further investigated by UNIPI and UNINA. *The multispectral image classification* developed at UNINA (see D4.FL3 section 2.2.8) will be considered as possible application to be implemented in FastFlow, with part of the parallelism exploited on shared memory multicore hw and part on a GP-GPU. Several versions of the application will be considered targeting embedded systems (e.g. NVidia Jetson nano style) to classical HPC systems (shared memory multicore clusters) to evaluate the possibilities offered in the HPC continuum perspective.

Support to the parallelisation of flagship demonstration

Within the demonstrator of section 2.1, aimed at providing HPC compression of huge collections of programs, the possibilities offered by FastFlow to a) implement parallel versions of the compression algorithm(s) on standard CPUs and b) orchestrate offloading of relevant part of the compression algorithm(s) to accelerators will be investigated. This activity involves UNIPI (different research groups) ENEA (the final user of the compression algorithms) and UNITO. In addition, INRIA (F) will be the source of the huge amount of data to be compressed.

2.10.5 Final validation tests

We plan to validate the activities related to FastFlow development and exploitation by:

- Being able to demonstrate the efficiency of the enhanced COW/NOW targeting support comparing the results achieved on HPC clusters by a FastFlow port of some microbenchmark application and those obtained by the original application developed using standard programming environments.
- Being able to offload computations to FaaS frameworks and to measure the differences in performance in beetween FaaS offloading and FastFlow only implementations.
- Being able to support and orchestrate parallel execution of different chunks of sequential business logic code in both the “compression” prototype of section 2.1, and in the execution of the multispectral image classification algorithm on stream of images in cooperation with UNINA [167,168]

2.11 Anomalous subgroup characterization with DivExplorer

2.11.1 Introduction

The rise in data availability and the prevalence of high-performance computing (HPC) have accelerated the advancement of machine learning (ML) and artificial intelligence (AI) models. ML models rely on substantial data and computing power for their training and optimization. The advent of HPC and big data technologies has enabled researchers and practitioners to process, store, analyze, deploy, and develop ML models with greater efficiency and effectiveness.
Nevertheless, with the increasing adoption of AI models, it is imperative to evaluate and ensure their quality and reliability.
The evaluation of ML model behavior generally focuses on overall performance, estimated over all the data. However, the overall estimation provides no indication if differences in the model behavior exist across subsets of data.
Models may perform differently on different data subgroups. The identification of these critical data subgroups plays an important role in many applications, for example, model validation and testing, model comparison, error analysis, or evaluation of model fairness.

When evaluating a model's performance, it is essential to understand its behavior across different subsets of data. By examining subgroup-specific performance, we can identify if the model's accuracy varies significantly across different demographic groups, conditions, or other relevant factors. This analysis helps ensure that the model is reliable and effective for all subgroups, not just the overall population.
Moreover, examining model behavior in subgroups helps in understanding the causes of errors or discrepancies in predictions. By identifying subgroups where the model consistently performs poorly, we can gain insights into the specific challenges or limitations faced by the model in those cases. This information can guide improvements in the model design or training process.

Typically, domain expert help is required to identify relevant (or sensitive) subgroups. Recent advances in ML model performance and analysis have recently proposed to address this task automatically [177,178,179,180]. The automation of the subgroup identification allows to efficiently explore and analyze subgroup performance. Among the pioneers in this line of work, we proposed DivExplorer [177], an automatic approach to explore datasets and find subgroups of data for which a model behaves in an anomalous manner. The notion of divergence is introduced to estimate the different classification behavior in data subgroups with respect to the overall behavior. Subgroups are characterized via patterns, defined as a set of attribute value, making the subgroups directly interpretable.

We envision a comprehensive framework to analyze the behavior of ML models, focusing on peculiarities at the subgroup level. The project, having as it cores component the DivExplorer approach, will cover multiple directions: (i) handling Big Data and Big Data models, (ii) generalize to multiple tasks and models, (iii) proposing novel methodologies for model comparison and selection, (iv) leverage subgroup analysis for model improvement and (iv) integration of subgroup analysis into interactive frameworks enabling access to computational data on a HPC system.

2.11.2 Related works

Subgroup analysis often rely on domain experts to identify the relevant (or sensitive) subgroups of interest. TensorFlow Model Analysis [181] and MLCube [182] belongs to this category. These approach enables interactive explorations and visualization, requiring the user to specify the subgroups of interests.
For fairness assessment, the diagnosis concentrates on evaluating if results are dependent on certain sensitive or protected attributes (e.g., gender, ethnicity, sexual orientation) [183,184].
Leveraging known or user-defined subgroups however requires human expertise and hinders the identification of unexpected and previously unknown critical subgroups.
Only recently, automatic subgroup detection techniques have been proposed to automatically identify subgroups with peculiar behavior. Works as Fairvis [185] adopt clustering techniques. However, the identified clusters are not directly interpretable, limiting the actionable understanding. Approaches as SliceLine e SliceFinder [179,180] leverage instead the notion of pattern as a conjunction of attribute-value pairs to slice the dataset. This allows a direct understanding of the conditions associated with a peculiar behavior. However, existing solutions adopt

heuristics to prune the search [179] or are optimized only to derive subgroups with lower performance than the average [180], not allowing for a complete understanding of the model behavior.
Existing approaches generally focus on discovering problematic subgroups in terms of classification or regression performance. We aim to propose a comprehensive framework that can be adopted to inspect the data and model behavior for generic functions, not limited to performance.
Moreover, a relevant component for understanding model behavior is the interactivity of subgroup analysis tool and the effective visualization of its results. Existing approaches address this requisite by proposing interactive web applications [178]. We plan an integration of the proposed subgroup exploration tool with the **Interactive Computing Service (IAC)** framework of CINECA partner, described in the DL5.FL3 section 2.1.4.

2.11.3 Actual prototype description and maturity level

The envisioned framework has as core component the DivExplorer approach. DivExplorer is currently at TRL 3 - Proof-of-Concept Demonstrated, Analytically and/or Experimentally.

a. Prototype modelization, structure and functional description

DivExplorer is an automatic approach to explore datasets and find subgroups of data for which a model behaves in an anomalous manner. The notion of divergence is introduced to estimate the different classification behavior in data subgroups with respect to the overall behavior. Subgroups are characterized via patterns, defined as a set of attribute values.
The approach algorithm is based on the effective integration of performance and divergence into the exploration process, leveraging frequent pattern mining algorithms. This enables DivExplorer to efficiently explore all subgroups with adequate representation in the dataset. Moreover, the use of the Shapley value and its generalization to analyze the contribution of the attribute value to the divergence has been introduced. The former allows understanding locally the contribution of each attribute value to the divergence of a specific subgroup. The latter allows understanding globally how much each attribute value contributes to the divergence of the model.
Differently from existing approaches, in this case efficient exploration of all subgroups with adequate representation in the dataset is allowed. As a result, the model behavior can be fully characterized. Furthermore, the proposed approach is model agnostic. Hence, it treats the classification model as a black box, without knowledge of its internal working.

b. Actual implementation

The subgroup analysis of DivExplorer is currently available in two versions: (i) python package of the pypi repository and (ii) web app.
The source code of DivExplorer is available as a python package. The core library it leverages are numpy, pandas, sklearn, numpy and mlextend for the exploration of subgroup.
The web app can be deployed don any cloud that provides services for running containerized web services. Currently, our hosting relies on Google Appengine. The back-end, which implements the data access

layers and analysis algorithms, is written in Python, and relies on the py4web web framework [186].
The analytical operations leverages the DivExplorer library, the Pandas library for dataset processing, and the scikit-learn library for data mining [9]. The front-end is written using the vue.js Javascript framework, which enables dynamic visualizations and explorations of the dataset. Data is stored in a cloud SQL database. In particular, we currently use Google Cloud Mysql and Google Cloud Storage.

c. Validation tests and results

The library can support any structured dataset. Domain experts and general users can use its open-source implementation to analyze the classification performance of generic machine-learning models for such data.
The current version of DivExplorer has been successfully applied to understand the behavior of ML classifiers on small- and large-scale data. The results are available in [177].
Moreover, the web app version of DivExplorer was demonstrated and discussed in paper [178].

2.11.4 Prototype evolution and implementation

a. Prototype evolution direction

The prototype evolution will cover the following directions.

Big data context. The efficient exploration of DivExplorer is suitable for parallel and distributed implementation, allowing its adoption in a Big Data context. The aim is to enhance it for understanding Big Data models. We aim to release a new implementation to allow its adoption in the Big Data context as a future work. We plan to release a new version to support running on Apache Hadoop/Spark clusters.

Task and model generalization. The current prototype has been successfully applied to understand the behavior of ML classifiers based on the sklearn library and tabular data. We plan to extend it and evaluate it to multiple tasks as regression, ranking, intent classification, automatic speech recognition and tasks and model architectures and implementations.
Model comparison and selection. The current prototype is adopted to understand the behavior of an individual ML model. We will extend the methodology to allow for the comparison of models at the subgroup level and the selection of the most suitable ones for the context under analysis.

Model improvement and iteration. The subgroup analysis allows gaining insight on problematic subgroup and could be a tool for model improvement. This may involve retraining the model with more diverse data, adjusting the model's algorithms or parameters, or implementing fairness-enhancing techniques. The iterative nature of the framework will allow for continuous improvement of the model's behavior in subgroups.

Interactive and accessible framework. A relevant aspect of the envision prototype is an easy and effortless interaction with end users. The prototype should easily integrate with a wide range of analysis and ML libraries, such as Pandas, NumPy, Scikit-learn, and PyTorch. Users should easily specify the data and the model to analyze and directly focus on the subgroup analysis enabled by our framework. We envision the integration of subgroup analysis into interactive frameworks enabling access to computational data on a HPC system. This would enable a seamless workflow, streamlining the analysis process and eliminating the need for switching between different tools or platforms.

b. Prototype evolution structure and description

The evolution of the prototype will cover the envisioned directions. Specifically, priority will be given to all the steps that would enable the integration (and its analysis of feasibility first) with the tools of other partners identified as relevant for the framework development.

c. Prototype Implementation and involved tools

We will investigate the integration and involvement into the proposed framework of the following tools: aMLLibrary (POLIMI), Interactive Computing Service – IAC (CINECA) provides and ParSoDA (UNICAL).

aMLLibrary. We will investigate the integration of DivExplorer subgroup analysis into the autoML solution of aMLLibrary (POLIMI). aMLLibrary is a Python package implements an autoML solution to train multiple regression models and automatically select the most accurate one based on the validation metric chosen. The analysis of performance at the subgroup could be a relevant component for the validation and the choice of the best regression model. The integration of DivExplorer and aMLLibrary would enable a comprehensive model comparison and selection for the regression task. See DL4.FL3 section 2.1.8

Interactive Computing Service – IAC. A relevant aspect of the envision prototype is a seamless workflow and an easy interaction with end users. We envision the integration of DivExplorer into the Interactive Computing Service – IAC of CINECA. ICA would enable access to computational data on a HPC system. This would enable a seamless workflow, streamlining the analysis process. The subgroup analysis functionality would be directly accessible in the Jupyter launcher of the IAC interface. By accessing the subgroup analysis capabilities via IAC, users can easily access and analyze relevant data, build ML models, and explore subgroup-specific behavior in a unified and cohesive manner.

ParSoDA. ParSoDA (Parallel Social Data Analytics) is a library that simplifies the development of parallel data mining applications executed on HPC systems. It provides a set of functions for processing and analyzing data. We will explore the integration of the BigData implementation of DivExplorer and ParSoDA (UNICAL) library for data analysis applications

2.11.5 Final validation tests

We will evaluate the prototype on a wide range on datasets, varying the dimensionality and cardinality. We will also test the approach on a wide range of tasks and architectures.
The evaluation will be based on qualitative evaluation and synthetic tests to evaluate the ability of the approach to capture model behavior at the subgroup level. Moreover, we will also consider user studies to evaluate the ability of the approach to provide insight into the model behavior and its interactivity level.

2.12 Compilation flow and deployment strategy targeting RISC-V accelerators for HPC computing

2.12.1 Introduction

In recent years, HPC and Cloud computing architectures are becoming increasingly complex since the growing demand for performance and energy efficiency has favored the proliferation of heterogeneous systems coupling standard processors with specialized accelerators [187,188]. Generating efficient executable code for these systems is one of the most complex tasks for software engineers, and compilation toolchains play a crucial role in providing techniques and methodologies to achieve optimal workload mapping. However, this scenario poses a severe challenge for efficient compiler design.

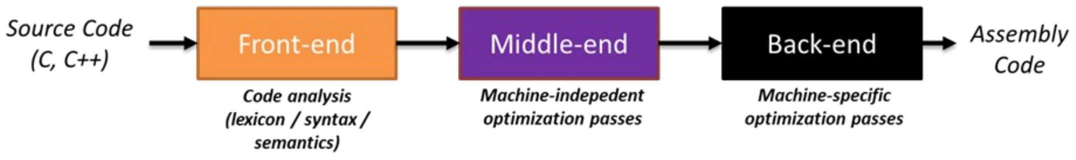


Figure 32: Main blocks composing a standard compiler toolchain

Figure 32 depicts the high level structure of a standard compiler toolchain [189], including three main stages: front-end, middle-end, and back-end. The front-end stage recognizes legal programs and produces an intermediate representation (IR) with an abstraction level suitable for the following transformations. Middle-end and back-end optimization passes transform an input program representation into an equivalent one optimized for a target metric (e.g., speed, size, or safety), and the design of the IR language must simplify this goal, adopting machine-independent or machine-specific knowledge, respectively. Even if this design is widespread and successfully applied in many application contexts, it is not flexible enough to target the heterogeneous scenario envisioned above.
In recent years, compiler researchers and companies have explored an approach base on multi-level intermediate representation (MLIR) [190]. Middle-end optimizations are generic by construction; for this reason, they cannot fully exploit the constraints provided by specific application domains. Conversely, back-end optimizations are fully focused on the features and peculiarities of the target machine. MLIR introduces a set of domain-specific middle-end representations (called dialects) geared toward domain-specific optimizations, allowing different levels of abstraction to co-exist freely using a uniform IR grammar.

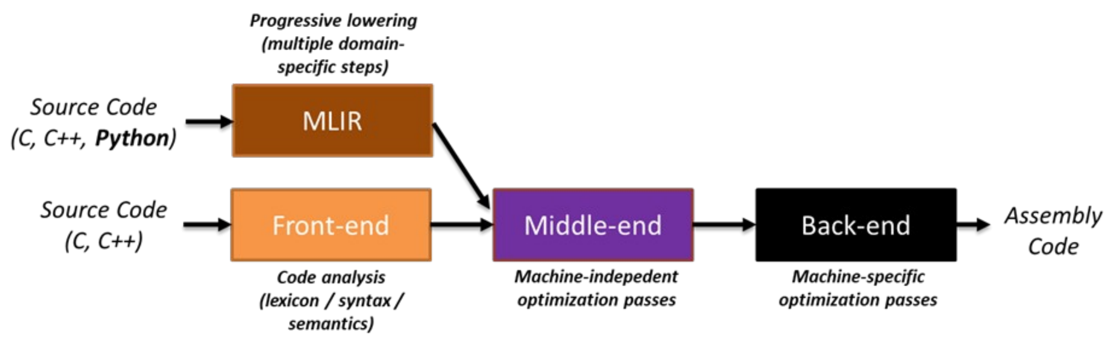

Progressive lowering (multiple domain-specific steps)
Source Code (C, C++, Python)
MLIR
Source Code (C, C++)
Front-end
Middle-end
Back-end
Assembly Code
Code analysis (lexicon / syntax / semantics)
Machine-indepedent optimization passes
Machine-specific optimization passes

Figure 33: Integration of MLIR features into a compiler toolchain

Figure 33 shows the integration of MLIR capabilities into a compiler toolchain. The MLIR tools intercept high-level program constructs with the aim to lower them progressively down to a low-level intermediate representation (usually the one used in the middle-end). This use case aims to demonstrate the MLIR flow into an HPC environment, providing support for high-level workloads targeting experimental RISC-V accelerators.

2.12.2 Related works

MLIR-enabled toolchains aim to lower the program progressively down to machine code, and are typically with a traditional compiler toolchain that performs the lowest-level transformations.

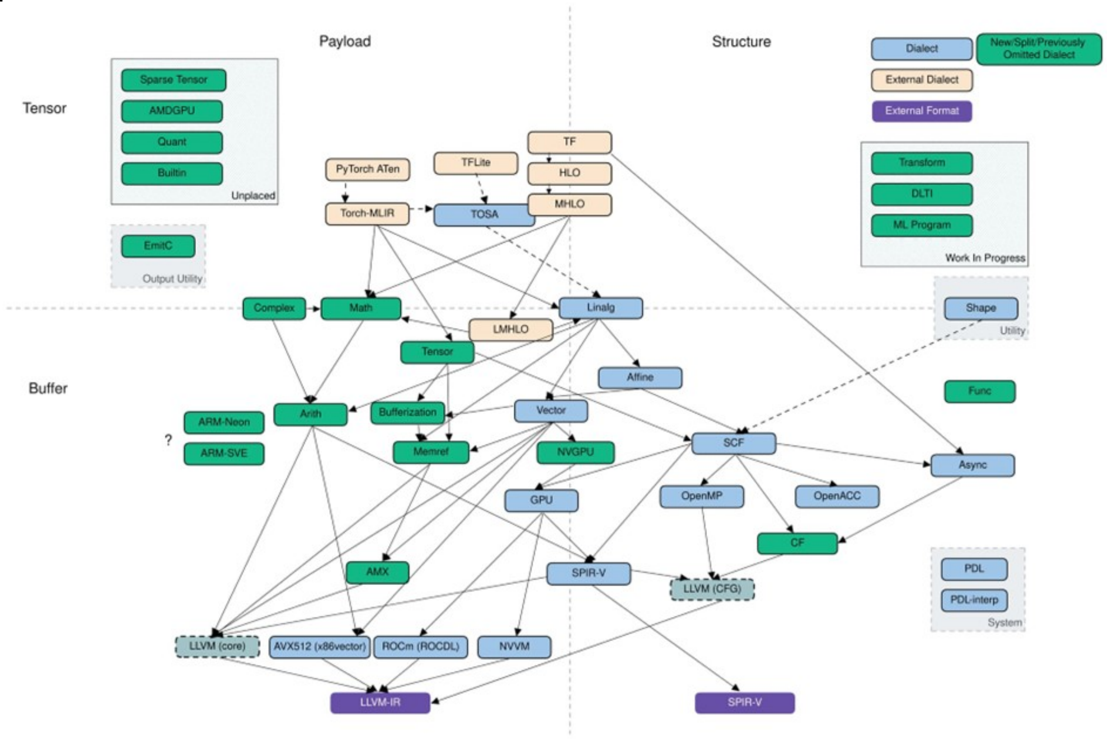


Figure 34: A graph depicting the current available MLIR dialects[194]

An optimization tool base on MLIR must schedule a set of transformation passes based on the available dialects [190]. A dialect is an IR language defining attributes, operations, and types; dialects are the most fundamental aspects of MLIR and can be used to model a variety of different abstractions (from arithmetic properties to pattern matching). Figure 34 shows a graph of the relations among the publicly available

dialects. It is noteworthy that all low-level dialects directly map on LLVM IR to ensure interoperability with the LLVM toolchain MLIR-based tools must schedule a path starting from the top (domain-specific dialects) down to the bottom (machine-specific dialects).
In recent years, many research contributions have focused on high-level dialects and have yet to provide an end-to-end solution[191, 192]. One of the most mature tools is TinyIREE [193], an MLIR-based toolchain to lower machine learning programs to mobile and edge devices. In our work, we want to realize a tool targeting HPC systems with a broader perspective, with no restriction to a single application domain and the possibility to investigate multiple lowering strategies. Moreover, we will add support to RISC-V accelerators specifically designed for HPC.

2.12.3 Actual prototype description and maturity level

The tool prototype is at an early development stage (TRL 2). We completed the specification and initial design of MLIR abstractions for a RISC-V accelerator for HPC computing, considering on a research prototype available as an open-source project[195].

a. Prototype modelization, structure and functional description

As discussed in the previous sections, the main goal of MLIR is to enable the lowering of high-level programs to low-level representations.

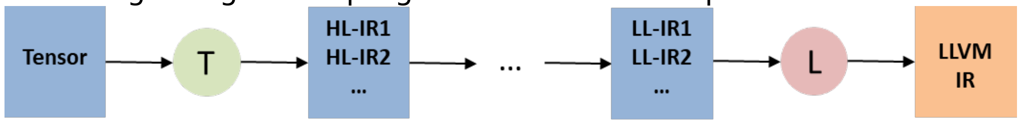


Figure 35: A compilation flow based on MLIR dialects (squares) and related transformations (circles).

Figure 35 shows a set of steps inside an MLIR optimization flow. The square shapes represent an intermediate program representation during the transformation flow, while the circles are transformations. The starting point is a single dialect at the highest level of abstraction (e.g., tensor); as a first step, it is translated into a new intermediate representation composed of a set of high-level dialects (HL-IRi), each one representing a different abstraction (e.g., math operations, loops, control flow). The last transformation is a lowering step from low-level dialects (LL-IRi) to the LLVM IR, which can be injected into an LLVM toolchain providing support for the target platforms.
Our tool will provide the implementation of the low-level dialects required to target a RISC-V accelerator and the related transformations from high-level dialects down to LLVM IR. It will also manage the orchestration of the optimization flow.

b. Actual implementation

We have completed the specification and started the design of the MLIR abstractions for RISC-V to map the hardware features in the ISA extensions supported by RISC-V accelerators, mainly hardware loops and auto-incremental data pointers. These low-level dialects are required to enable the adoption of these features in the MLIR optimization toolchain, whose upper levels are platform-agnostic thanks to these abstractions.

c. Validation tests and results

At the current development stage, we have tested the lowering of a generalized matrix multiplication kernel (GEMM) from the high-level representation down to the low-level dialects that we have designed, and we have started testing its integration with the LLVM environment.

2.12.4 Prototype evolution and implementation

At the end of the project, we plan to reach a TRL equal to 5. The key performance indicators (KPIs) that we will consider for the prototype evaluation are the following:

- Lines of code and level of abstraction of the input program.
- Performance of the compiled program compared to hand-tuned code.
- Ease of use for the programmer compared to a standard compiler toolchain

a. Prototype evolution direction

The prototype evolution will follow an iterative development approach, with the aim to integrate multiple tools into an MLIR-based compilation flow and demonstrate the approach on a RISC-V accelerator using a simulation environment.

b. Prototype evolution structure and description

The prototype evolution will follow these steps:

1. Integration of the low-level dialects with the LLVM [196] toolchain.
2. Adoption of a tool to generate the highest-level dialect from a widely-used programming language (e.g., Python).
3. Definition of a set of workflows for at least two relevant scenarios (deep neural networks and linear algebra workloads).
4. Adoption of a workflow management tool to orchestrate the MLIR transformation steps and explore different solutions.

c. Prototype Implementation and involved tools

Following the prototype evolution stages, we plan to integrate three main tools:

- **mlir-opt** (distributed as a component of the LLVM compilation toolchain): This tool will manage transformation between MLIR dialects and produces an intermediate format for the LLVM toolchain (LLVM IR). The MLIR based approach is one of the contributions described in DF4.FL3 section 2.5.1.
- **xDSL**: https://github.com/xdslproject/xdsl This tool generates MLIR from high-level code (Python) and simplyfies the description of high-level dialects/transformations.
- **StreamFlow (UNITO)**: This tool will orchestrate the transformations steps by invoking mlir-opt with different parameters. This tool in introduced in DF4.FL3 section 2.1.6

2.12.5 Final validation tests

As final validation tests, we will evaluate our prototype in a large-scale multi-cloud environment to verify the feasibility of the approach on real-world case studies. As part of this validation, we would like to show how the desired prototype can implement continuous re-allocation of computing resources in many computing clusters and to adapt to the current workload or environmental conditions.

2.13 National Federated Cloud/HPC Infrastructure

2.13.1 Introduction

Current open-source orchestrators for cloud/HPC infrastructures (e.g., Kubernetes in its multiple flavors, OpenStack) handle each infrastructure as a multitude of (connected) isolated silos instead of a unique virtual space. This leads to a sub-optimal fragmented view of the overall available resources, preventing the seamless deployment of fully distributed applications, or the usage of existing applications installed in another cluster, but operating on remote data present in the current cluster (i.e., data-gravity approach).
Current federated approaches represent a partial solution to this problem because the federation among participating clusters has to be established a-priori and it is rigid in its nature, hence difficult to extend/resize dynamically. Furthermore, most of the existing approaches still partition the whole federation in multiple sites (i.e., the clusters contributing to the federation), with little (or no) primitives facilitating the communication among distributed applications.
Liqo offers a possible answer to this problem, enabling (1) highly dynamic federation mechanisms, which (2) can be set up/torn down in a matter of seconds, and (3) provides services, networking and storage transparency to all the applications running in any of the federated clusters, hence transforming any multi-cluster deployment into a single (virtual) cluster one.
The current project aims at creating a dynamic federation among participating Italian actors (e.g., Universities, research organizations), using Liqo.io, enabling the sharing of computing and storage resources, applications and services, as well as data, among all the involved organization.

2.13.2 Related works

The most common approach to cloud federation, as far as Kubernetes is concerned, is the KubeFed [197] project, which creates a federation among Kubernetes cluster. However, this approach is rigid, and it has been considered not appropriated even from the Kubernetes community itself. A potential successor is the Karmada project [198], funded by a group of Chinese companies, but it is not compatible with Vanilla Kubernetes and it requires users to rely on a new API to control the cluster.
Other approaches, such as Submariner [199] or Cilium Mesh [200] are limited to networking transparency, hence providing full connectivity among clusters but lacking services and data transparency across the federated cluster. It is worth nothing that those approaches are compatible with the latest versions of Liqo, hence enabling a mixed solution in which Liqo takes care of handling Kubernetes resources (pods, services,

volumes), while the network connectivity is provided by the former projects.
Finally, other approaches such as SUSE Rancher Fleet [201] or KubeEdge [202] provide an overarching infrastructure to control multiple clusters from a single pane of glass, but without the transparency (computing, storage, services) that is required for seamless deployment of multi-cluster applications.
With respect to OpenStack [203], this software offers federation primitives though proper extensions, but this should be considered more as a collection of different clusters controlled by a single entity than a geographical (transparent) infrastructure.
In a nutshell, Liqo is the only project that provides full transparency with respect to computing, networking and storage resources, as well as Kubernetes services and primitives, among all the entities participating in the virtual cluster. However, Liqo has been designed to operate in Kubernetes environments and currently it does not provide support for “legacy” virtualization primitives such as VMs.

2.13.3 Actual prototype description and maturity level

Liqo is an open-source project that enables dynamic and seamless Kubernetes multi-cluster topologies, supporting heterogeneous on-premises, cloud and edge infrastructures, providing the following features:

- **Peering**: Automatic peer-to-peer establishment of resource and service consumption relationships between independent and heterogeneous clusters, with an automatic and transparent VPN between involved clusters.
- **Offloading**: Seamless workloads offloading to remote clusters, without requiring any modification to Kubernetes or the applications themselves. Multi-cluster is made native and transparent, hence enabling to collapse an entire remote cluster to a virtual node that is compliant with standard Kubernetes approaches and tools.
- **Network Fabric**: A transparent network fabric, enabling multi-cluster pod-to-pod and pod-to-service connectivity, regardless of the underlying configurations and networking plugins. Users can natively access the services exported by remote clusters, and spread interconnected application components across multiple infrastructures, with all cross-cluster traffic flowing through secured network tunnels.
- **Storage Fabric**: A native storage fabric, supporting the remote execution of stateful workloads according to the data gravity approach. It enables the seamless extension of standard (e.g., database) high availability deployment techniques to the multi-cluster scenarios, for increased guarantees, without the complexity of managing multiple independent cluster and application replicas.

An example of Liqo running on a real site (through its minimal dashboard) is available at [204].
The current maturity level of the software is TRL 5, with the software already being in production (although experimental) at Politecnico di Torino, as described in this blog post [205] and being under testing by independent partners, such as the Dutch Gaia-X community, as described in this press news[206].

a. Prototype modelization, structure and functional description

Liqo does not introduce any modification in standard Kubernetes APIs for application deployment and well-established management workflows, as well as to support a wide range of common infrastructures, with no constraints in terms of cluster type (i.e., on-premises or hosted by a cloud provider) and networking configurations (i.e., network providers and IP addresses).
Liqo leverages the virtual node concept to masquerade the resources shared by each remote cluster. This solution allows the transparent extension of the local cluster, with the new capabilities seamlessly taken into account by the vanilla Kubernetes scheduler when selecting the best place for the workload's execution. The virtual node abstraction is implemented through an extended version of the Virtual Kubelet project (https://github.com/virtual-kubelet/virtualkubelet). In Kubernetes, the kubelet is the primary node agent, responsible for registering the node with the control plane and handling the lifecycle of the pods (i.e., the minimum scheduling unit, composed of one or many containers sharing the same network namespace) assigned to that node. The virtual kubelet (VK) replaces a traditional kubelet when the controlled entity is not a physical node, allowing to control arbitrary objects through standard Kubernetes APIs. Hence, it enables custom logic to handle the lifecycle of both the node itself and the pods therein hosted.
A simple overview of the Liqo architecture is depicted in figure 36.

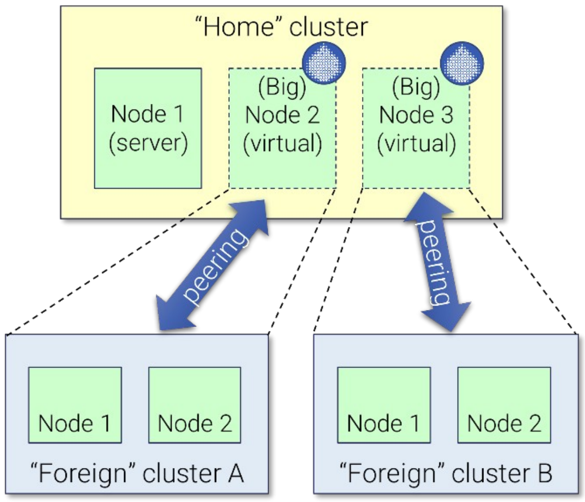


Figure 36: overview of the Liqo architecture

b. Actual implementation

The current implementation (available at [207]) supports multiple Kubernetes implementations and flavors (Kubernetes vanilla, OpenShift, KinD, K3s), including the ones running on major cloud providers (Amazon EKS, Azure AKS, Google GKE), and the major CNI network providers (e.g., Calico, Flannel, Cilium). The software is entirely developed in Golang, with an easy-to-setup install, and it also includes a minimal dashboard to control the peering process, i.e., the procedure required to establish a sharing resource session between two clusters.
More details about the current implementation and internals are available on the official documentation page[208].

c. Validation tests and results

The current implementation has been proved not to introduce no noticeable penalty compared to vanilla Kubernetes, even in novel multi-cluster and multi-cloud contexts. In other words, Liqo introduces the support for Kubernetes to operate in new deployment scenarios, without any performance penalty. Detailed results are presented in the paper [209].
In addition, this software has been used to provide production-quality services in several real environments, such as in Politecnico di Torino and TNO. More details on the Liqo blog page [210].

2.13.4 Prototype evolution and implementation

a. Prototype evolution direction

The current prototype has been used mainly either in cloud-to-cloud scenarios, or on on-premises-to-cloud. In both cases, a single actor can be considered the owner of the entire infrastructure. This leaves several options for the future directions of the prototype.

- **Security and multi-tenancy**: they represent fundamental features when the clusters that share resources together belong to different actors. In this case, well-defined boundaries must be created that separate the jobs of the two actors, even running on the same cluster. In addition, primitives for secure execution of workloads (e.g., no tampering) are needed to provide better guarantees that running workloads cannot be inspected by the cluster owner.
- **Scalability**: When the network infrastructure becomes bigger, e.g., including the resources of several federated clusters, new scalability levels should be achieved that go beyond the well-known scalability properties of a Kubernetes cluster.
- **Edge clusters**: this novel deployment scenario, consisting in one (or more, for redundancy reasons) master cluster coordinating the operations of a multitude of geographically dispersed clusters, running at the edge of the network, represents a new, challenging deployment scenario. This includes the necessity to address severe scalability constraints, as well as geographical limitations (e.g., network bandwidth, latency, resource heterogeneity).

The current prototype, running at TRL-5, will be extended with the features targeting the above directions, with a final outcome over the course of the project that targets TRL 6, including the new features.

b. Prototype evolution structure and description

The planned evolution of the prototype is oriented to the creation of a vibrant community of enthusiast cloud managers who are willing to share computing resources and data across Italian institutions. The biggest effort planned in this activity is to gather interests of the potential involved actors in order to create, on a voluntary base, the consensus for an highly-dynamic nationwide federation of cloud and HPC resources, available on-demand, which is (a) able to challenge major cloud hyperscalers in terms of available resources and software services, which are specifically designed for academic/scientific experimentation, and (b)

contributes to lower the actual cost of massive scientific computing-based experiments due to the resource sharing among involved partners.
A possible example of an initial federation of cloud resources is available at [8].

c. Prototype Implementation and involved tools

One of the biggest challenges in the above vision is the necessity to integrate, with the Liqo technology, tools and methodologies that can complement its features, in particular (a) the capability to predict the performance of the federated infrastructure, and (b) the capability to control infrastructures that are not fully cloud-native (e.g., Kubernetes). For this reasons, two tools are considered the best options:

- **BDMaaS+ (UNIFE)**: it implements Digital Twin methodologies to enable an accurate representation of applications operating in multi-cloud and cloud continuum scenarios. For creating this virtual representation, BDMaaS+ makes use of input static description of an application (TOSCA blueprint) and the state of resources available across the multi-cloud. By capturing the state of an existing HPC application through a virtual representation of the HPC application it would be possible to run simulation-based accelerated timescale analysis and to select a proper deployment description. See D4.FL3 section 2.1.1.
- **INDIGO(UNIBO):** it is able to find the most appropriate set of computing resources considering the application requirements, application provider defined policies (pricing, latency), and the current availability of resources among the multi-cloud, and it can translate high-level intents (e.g., TOSCA) into proper infrastructure-based commands that can be used to drive the actual Liqo federation and the actual resources to be used in a specific deployment. See D4.FL3 section 2.3.2.

2.13.5 Final validation tests

We plan to test the tool with at least two applications in the domains of deep neural networks and linear algebra. Since the target hardware platforms are experimental and not available as commercial solutions, we will target a simulator on a Docker environment. Our goal is to start from high-level code and obtain performance results comparable to hand-tuned applications (at least 90% of the maximum performance achievable).

3 References

[1] T. Hey, S. Tansley, J. Gray, and K. Tolle, “The fourth paradigm: data-intensive scientific discovery.”, *Microsoft research*, 2009

[2] V. Springel, N. Yoshida, and S. D. M. White, “GADGET: a code for collisionless and gasdynamical cosmological simulations.”, *New Astronomy*, p. 79-117, 2001

[3] P. E. Dewdney, P. J. Hall, R. T. Schilizzi, and T. J. L. Lazio, “The square kilometre array.”, in *Proceedings of the IEEE*, vol. 97, no. 8, p. 1482-1496, 2009

[4] E. Sciacca and et al., "An integrated visualization environment for the virtual observatory: Current status and future directions.", *Astronomy and*

Computing, vol. 11, p. 146-154, 2015

[5] VisIVOLab GitHub repository, “VisIVO Server”, [Online]. Available: https://github.com/VisIVOLab/VisIVOServer

[6] VisIVOLab GitHub repository, “ViaLactea Visual Analytics”, [Online]. Available: https://github.com/VisIVOLab/ViaLacteaVisualAnalytics

[7] F. Vitello and et al., “Vialactea visual analytics tool for star formation studies of the galactic plane.”, *Publications of the Astronomical Society of the Pacific*, vol. 130, no. 990, 2018

[8] Kitware, “The Visualization Toolkit (VTK)”, [Online]. Available: https://vtk.org/

[9] E. Sciacca and et al., “Scientific Visualization on the Cloud: the NEANIAS Services towards EOSC Integration.”, *Journal of Grid Computing*, vol. 20, no. 7, 2022

[10] OpenACC Organization, “OpenACC”, [Online]. Available: https://www.openacc.org/

[11] K. Moreland, C. Sewell, W. Usher, L. T. Lo, J. Meredith and et al., “Vtk-m: Accelerating the visualization toolkit for massively threaded architectures.”, IEEE computer graphics and applications, p. 48-58, 2016

[12] CWL Project, “Common Workflow Language (CWL)”, [Online]. Available: https://www.commonwl.org/

[13] The University of Manchester UK and HITS gGmbH, “Workflow Hub”, [Online]. Available: https://workflowhub.eu/

[14] University of Technology Sydney, The University of Manchester UK and RO-Crate contributors, “RO-CRATE”, [Online]. Available: https://www.researchobject.org/ro-crate/

[15] The Galaxy Community, “Galaxy”, [Online]. Available: https://galaxyproject.org/

[16] The Apache Software Foundation, “Airflow”, [Online]. Available: https://airflow.apache.org/

[17] L. Franke and D. Haehn, “Modern scientific visualizations on the web”, *Informatics*, 2020

[18] M. Raji, A. Hota, T. Hobson and J. Huang, “Scientific visualization as a microservice.”, *IEEE Transaction on Visualization and Computer Graphics*, vol. 26, no. 4, p. 1760–1774, 2018

[19] J. Ahrens, B. Geveci and C. Law, “Paraview: An end-user tool for large data visualization.” *The Visualization Handbook*, 2005

[20] T. Goodale, G Allen, G. Lanfermann, J. Masso, T. Radke, E. Seidel and J. Shalf, “The cactus framework and toolkit: Design and applications.”, *Proceedings of the 5th international conference on High performance computing for computational science*, p. 197–227, 2002

[21] Y. Zhou, R. M. Weiss, E. McArthur, D. Sanchez, X. Yao, D. Yuen, M. R. Knox and W. W. Czech, “Webviz: A web-based collaborative interactive visualization system for large-scale data sets”, *GPU Solutions to Multi-scale Problems in Science and Engineering. Lecture Notes in Earth System Sciences*, p. 587–606, 2013

[22] U. Becciani, E. Sciacca, A. Costa, P. Massimino, C. Pistagna and et al., “Science gateway technologies for the astrophysics community.” *Concurrency and Computation: Practice and Experience*, vol. 27, no. 2, p. 306–327, 2015

[23] VisIVO Team, “VisIVO”, [Online]. Available: https://visivo.readthedocs.io/

[24] Z. Ahmed, “Practicing precision medicine with intelligently integrative clinical and multi-omics data analysis.”, *Human Genomics*, vol. 14, no. 1, p. 35, 2020

[25] S. Goodwin, J. D. McPherson, and W. R. McCombie, “Coming of age: ten years of next-generation sequencing technologies,” *Nature Reviews Genetics*, vol. 17, no. 6, p. 333–351, 2016

[26] Y. Wang, Y. Zhao, A. Bollas, Y. Wang, and K. F. Au, “Nanopore sequencing technology, bioinformatics and applications,” *Nature Biotechnology*, vol. 39, no. 11, p. 1348–1365, 2021

[27] H. Li et al., “The Sequence Alignment/Map format and SAMtools,” *Bioinformatics*, vol. 25, no. 16, pp. 2078–2079, 2009

[28] A. McKenna et al., “The Genome Analysis Toolkit: a MapReduce framework for analyzing next-generation DNA sequencing data.,” *Genome Res*, vol. 20, no. 9, pp. 1297–1303, 2010.

[29] R. Poplin et al., “A universal SNP and small-indel variant caller using deep neural networks,” *Nature Biotechnology*, vol. 36, no. 10, pp. 983–987, 2018

[30] Kyle A. O’Connell et al., “Accelerating genomic workflows using NVIDIA Parabricks,” *bioRxiv*, 2022

[31] P. Di Tommaso, M. Chatzou, E. W. Floden, P. P. Barja, E. Palumbo, and C. Notredame, “Nextflow enables reproducible computational workflows,” *Nature Biotechnology*, vol. 35, no. 4, pp. 316–319, 2017.

[32] P. Cingolani et al., “A program for annotating and predicting the effects of single nucleotide polymorphisms, SnpEff,” *Fly*, vol. 6, no. 2, pp. 80–92, 2012

[33] K. Wang, M. Li, and H. Hakonarson, “ANNOVAR: functional annotation of genetic variants from high-throughput sequencing data,” *Nucleic Acids Research*, vol. 38, no. 16, pp. e164–e164, 2010

[34] J. G. Tate et al., “COSMIC: the Catalogue Of Somatic Mutations In Cancer,” *Nucleic Acids Research*, vol. 47, no. D1, pp. D941–D947, 2019

[35] Q. Li and K. Wang, “InterVar: Clinical Interpretation of Genetic Variants by the 2015 ACMG-AMP Guidelines,” *The American Journal of Human Genetics*, vol. 100, no. 2, pp. 267–280, 2017

[36] M. J. Landrum *et al.*, “ClinVar: public archive of relationships among sequence variation and human phenotype,” *Nucleic Acids Research*, vol. 42, no. D1, pp. D980–D985, 2014

[37] I. Colonnelli, B. Cantalupo, R. Esposito, M. Pennisi, C. Spampinato, and M. Aldinucci, “HPC Application Cloudification: The StreamFlow Toolkit,” in *Open Access Series in Informatics (OASIcs)*, p. 5:1-5:13, 2021

[38] D. Lan, R. Tobler, Y. Souilmi, and B. Llamas, “Genozip: a universal extensible genomic data compressor,” *Bioinformatics*, vol. 37, no. 16, pp. 2225–2230, 2021

[39] S. Deorowicz, A. Danek, and M. Kokot, “VCFShark: how to squeeze a VCF file,” *Bioinformatics*, vol. 37, no. 19, pp. 3358–3360, 2021,

[40] F. H. Cabrini, F. Valiante Filho, P. Rito, A. Barros Filho, S. Sargento, A. Venancio Neto and S. T. Kofuji, “Enabling the industrial internet of things to cloud continuum in a real city environment”, *Sensors 21*, 2021

[41] P. Beckman, J. Dongarra, N. Ferrier, G. Fox, T. Moore, D. Reed and M. Beck, “Harnessing the computing continuum for programming our world”, *Fog Computing: Theory and Practice*, p. 215–230, 2020

[42] D. Balouek-Thomert, E. G. Renart, A. R. Zamani, A. Simonet and M. Parashar, “Towards a computing continuum: Enabling edge-to-cloud integration for data-driven workflows”, *The International Journal of High Performance Computing Applications,* no. 33, p. 1159–1174, 2019

[43] M. AbdelBaky, M. Zou, A. R. Zamani, E. Renart, J. Diaz-Montes and M. Parashar, “Computing in the continuum: Combining pervasive devices and services to support data-driven applications” in *Proceedings of IEEE 37th International Conference on Distributed Computing Systems*, pp. 1815–1824, 2017

[44] S. Risco, G. Molt´o, D. M. Naranjo and I. Blanquer, “Serverless workflows for containerised applications in the cloud continuum”, *Journal of Grid Computing*, no. 19, p. 1–18, 2021

[45] H. Shafiei, A. Khonsari and P. Mousavi, “Serverless computing: a survey of opportunities, challenges, and applications”, *ACM Computing Surveys*, no. 54, p. 1–32, 2022

[46] J. Menetrey, M. Pasin, P. Felber and V. Schiavoni, “Webassembly as a common layer for the cloud-edge continuum”, in *Proceedings of the 2nd Workshop on Flexible Resource and Application Management on the Edge*, p. 3–8, 2022

[47] The Linux Foundation, “Kubernetes”, [Online]. Available: https://kubernetes.io/

[48] A. Haas, A. Rossberg, D. L. Schuff, B. L. Titzer, M. Holman, D. Gohman, L. Wagner, A. Zakai and J. Bastien, “Bringing the web up to speed with webassembly”, in *Proceedings of the 38th ACM SIGPLAN Conference on Programming Language Design and Implementation*, p. 185–200, 2017

[49] T. Lynn, J. G. Mooney, B. Lee and P. T. Endo, “The cloud-to-thing continuum: opportunities and challenges in cloud, fog and edge computing”, *Springer Nature*, 2020.

[50] J. Santos, T. Wauters, B. Volckaert and F. De Turck, “Towards low-latency service delivery in a continuum of virtual resources: State-of-the-art and research directions”, *IEEE Communications Surveys & Tutorials*, no. 23, p. 2557–2589, 2021

[51] L. Bittencourt, R. Immich, R. Sakellariou, N. Fonseca, E. Madeira, M. Curado, L. Villas, L. DaSilva, C. Lee and O. Rana, “The internet of things, fog and cloud continuum: Integration and challenges”, *Internet of Things*, no. 3-4, p. 134–155, 2018

[52] HiPEAC, “Vision 2021”, [Online]. Available: https://www.hipeac.net/vision/#/latest/articles/?q=continuum, 2021.

[53] S. Latre, J. Famaey, F. De Turck and P. Demeester, “The fluid internet: service-centric management of a virtualized future internet”, *IEEE Communications Magazine*, no. 52, p. 140–148, 2014

[54] M. Villari, M. Fazio, S. Dustdar, O. Rana and R. Ranjan, “Osmotic computing: A new paradigm for edge/cloud integration”, *IEEE Cloud Computing*, no. 3, p. 76–83, 2016.

[55] A. Camero and E. Alba, “Smart city and information technology: A review”, *Cities*, no. 93 , p. 84–94, 2019

[56] B. Neha, S. K. Panda, P. K. Sahu, K. S. Sahoo and A. H. Gandomi, “A systematic review on osmotic computing”, *ACM Transactions on Internet of Things*, no. 3, p. 1–30, 2022

[57] P. K. Gadepalli, S. McBride, G. Peach, L. Cherkasova and G. Parmer, "Sledge: a serverless-first, light-weight wasm runtime for the edge", in *Proceedings of the 21st International Middleware Conference*, p. 265–279, 2020

[58] A. Hall and U. Ramachandran, "An execution model for serverless functions at the edge", in *Proceedings of the International Conference on Internet of Things Design and Implementation*, p. 225–236, 2019

[59] S. Shillaker and P. Pietzuch, "Faasm: lightweight isolation for efficient stateful serverless computing", in *2020 {USENIX} Annual Technical Conference*, p. 419–433, 2020

[60] M. Malawski, A. Gajek, A. Zima, B. Balis and K. Figiela, "Serverless execution of scientific workflows: Experiments with hyperflow, AWS lambda and Google Cloud functions", *Future Generation Computer Systems*, no. 110, p. 502–514, 2020

[61] P. Bellavista and A. Zanni, "Feasibility of fog computing deployment based on docker containerization over Raspberrypi", in *Proceedings of the 18th international conference on distributed computing and networking*, p. 1–10, 2017

[62] R. Fielding, "Representational state transfer (REST)", [Online]. Available: https://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm, 2000.

[63] A. Langley, A. Riddoch, A. Wilk, A. Vicente, C. Krasic, D. Zhang, F. Yang, F. Kouranov, I. Swett, J. Iyengar, et al., "The Quic transport protocol: Design and internet-scale deployment", in *Proceedings of the conference of the ACM special interest group on data communication*, p. 183–196, 2017

[64] Cloud Foundry, "Open service broker", [Online]. Available: https://www.openservicebrokerapi.org/, 2016.

[65] C. Bormann, A. P. Castellani and Z. Shelby, "Coap: An application protocol for billions of tiny internet nodes", *IEEE Internet Computing,* no. 16, p. 62–67, 2012

[66] Deis Labs, "Akri", [Online]. Available: https://github.com/deislabs/akri, 2021

[67] AWS Open Source Blog, "Why AWS loves Rust and how we'd like to help", [Online]. Available: https://aws.amazon.com/blogs/opensource/why-aws-loves-rust-and-how-wed-like-to-help/, 2020

[68] M. Jacobsson and J. Willen, "Virtual machine execution for wearables based on webassembly", in *Proceeding of 13th EAI International Conference on Body Area Networks*,, pp. 381–389, 2020

[69] G. Peach, R. Pan, Z. Wu, G. Parmer, C. Haster and L. Cherkasova, "ewasm: Practical software fault isolation for reliable embedded devices", *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, no. 39, p. 3492–3505, 2020

[70] Bytecode Alliance, "wasmtime", [Online]. Available: https://github.com/bytecodealliance/wasmtime, 2021.

[71] wasm3, "wasm3 performance", [Online]. Available:https://github.com/wasm3/wasm3/blob/main/docs/Performance.md, 2021.

[72] OCI, "Artifacts",[Online]. Available: https://github.com/opencontainers/artifacts/blob/main/artifact-authors.md, 2022.

[73] T. Yuki, "Understanding polybench/c 3.2 kernels", in *International workshop*

on Polyhedral Compilation Techniques (IMPACT), pp. 1–5, 2014

[74] Parity, “wasmi”, [Online]. Available: https://github.com/paritytech/wasmi, 2021.

[75] Deis Labs, “Krustlet”, [Online]. Available: https://github.com/deislabs/krustlet, 2021.

[76] Rancher, k3s, [Online]. Available: https://k3s.io/, 2021.

[77] Node.js, “The node.js event loop”, [Online]. Available: https://nodejs.dev/learn/the-nodejs-event-loop, 2021

[78] Deislabs, “wasi-experimental-http”, [Online]. Available: https://github.com/deislabs/experimental-http, 2022

[79] U. Manber, "Finding Similar Files in a Large File System," in *USENIX Winter 1994 Technical Conference*, 1994.

[80] N. Heintze, "Scalable Document Fingerprinting," in *USENIX Workshop on Electronic Commerce, 1996*, 1996.

[81] A. Z. Broder, "On the resemblance and containment of documents," in *Proceedings of Compression and Complexity of SEQUENCES*, 1997.

[82] M. Charikar, "Similarity estimation techniques from rounding algorithms," in *Proceedings of ACM Symposium on Theory of Computing*, 2002.

[83] M. Frobe and alii, "CopyCat: Near-Duplicates Within and Between the ClueWeb and the Common Crawl," in *Proceedings of the ACM SIGIR Conference on Research and Development in Information Retrieval*, 2021.

[84] G. Singh Manku, A. Jain and A. Das Sarma, "Detecting near-duplicates for web crawling," in *Proceedings of the International Conference on World Wide Web*, 2007.

[85] M. R. Henzinger, "Finding near-duplicate web pages: a large-scale evaluation of algorithms," in *Proceedings of the ACM SIGIR Conference on Research and Development in Information Retrieval*, 2003.

[86] S. Schleimer, D. Shawcross Wilkerson and A. Aiken, "Winnowing: Local Algorithms for Document Fingerprinting," in *Proceedings of the ACM SIGMOD Conference on Management of Data*, 2003.

[87] P. Ferragina and G. Manzini, "On compressing the textual web," in *Proceedings of the International Conference on Web Search and Web Data Mining*, 2010.

[88] M.A. Zaharia, A. Ghodsi, R. Xin, and M. Armbrust. “Lakehouse: A new generation of open platforms that unify data ware-housing and advanced analytics”. In *Proceedings of the Conference on Innovative Data Systems Research*, 2021.

[89] Inria, “Software Heritage Archive”, [Online]. Available: https://www.softwareheritage.org/

[90] L. Belcastro, F. Marozzo, D. Talia and P. Trunfio, "ParSoDA: High-Level Parallel Programming for Social Data Mining". *Social Network Analysis and Mining*, vol. 9, no. 1, 2019

[91] Jupyter consortium ,“Project Jupyter”, [Online]. Available: https://jupyter.org/

[92] Red Hat Inc., “Ansible”, [Online]. Available: https://www.ansible.com/

[93] Anaconda Inc., “Anaconda”, [Online]. Available: https://www.anaconda.com/

[94] QuantStack, “Mamba package manager”, [Online]. Available: https://github.com/mamba-org/mamba

[95] Jupyter consortium, “Jupyter server proxy”, [Online]. Available:https://github.com/jupyterhub/jupyter-server-proxy

[96] Jupyter consortium, “Jupyter xeus”, [Online]. Available: https://github.com/jupyter-xeus/xeus

[97] Julia consortium, “Julia programming language”, [Online]. Available:https://julialang.org/

[98] R foundation, “R project ”, [Online]. Available: https://www.r-project.org/

[99] Microsoft Inc., “Visual Studio”, [Online]. Available:https://code.visualstudio.com/

[100] E4 COMPUTER ENGINEERING S.p.A, “E4 computer engineering”, [Online]. Available:https://www.e4company.com/

[101] E4 COMPUTER ENGINEERING S.p.A, “GAIA: GPU Appliance per l’Intelligenza Artificiale”, [Online]. Available:https://www.e4company.com/intelligenza-artificiale-e-gpu-appliance/

[102] Fenix Research infrastructure, “Interactive Computing E-Infrastructure for the Human Brain Project”, [Online]. Available:https://cordis.europa.eu/project/id/800858

[104] The VirtualGL Project, “TurboVNC”, [Online]. Available:https://www.turbovnc.org

[105] Jupyter consortium, “Jupyter Xpra”, [Online]. Available:https://github.com/FZJ-JSC/jupyter-xprahtml5-proxy

[106] R foundation, “R Studio”, [Online]. Available: https://www.r-studio.com/

[107] GNU, “Octave programming language”, [Online]. Available:https://octave.org/

[108] CINECA, “Leonardo super computer”, [Online]. Available:https://leonardo-supercomputer.cineca.eu/

[109] OpenFaaS Limited, “OpenFaaS”, [Online]. Available:https://www.openfaas.com, 2023.

[110] Apache foundation, “Apache OpenWhisk”, [Online]. Available:https://openwhisk.apache.org, 2023.

[111] S. Shillaker and P. Pietzuch, "Faasm: Lightweight isolation for efficient stateful serverless computing", *Proc. of 2020 USENIX Annual Technical Conf. (ATC ’20)*, 2020

[112] P. K. Gadepalli, S. McBride, G. Peach, L. Cherkasova, G. Parmer, "Sledge: A serverless-first, light-weight wasm runtime for the edge", *Proc. of Middleware ’20*, 2020.

[113] X. Lyu, L. Cherkasova, R. Aitken, G. Parmer, T. Wood, "Towards efficient processing of latency-sensitive serverless DAGs at the edge", *Proc. of 5th ACM Int’l Workshop on Edge Systems, Analytics and Networking (EdgeSys ’22)*, 2022.

[114] F. Lordan, D. Lezzi, R. M. Badia, "Colony: Parallel functions as a service on the cloud-edge continuum", *Proc. of Euro-Par ’21*, 2021.

[115] T. Pfandzelter, D. Bermbach, "tinyfaas: A lightweight faas platform for edge environments", *Proc. of 2020 IEEE Int’l Conf. on Fog Computing (ICFC ’20)*, 2020.

[11 A. Das, A. Leaf, C. A. Varela, S. Patterson, "Skedulix: Hybrid cloud scheduling

[116] for cost-efficient execution of serverless applications", *Proc. of IEEE CLOUD '20*, 2020.

[117] M. Ciavotta, D. Motterlini, M. Savi, A. Tundo, "DFaaS: Decentralized function-as-a-service for federated edge computing", *Proc. of CloudNet '21*, 2021.

[118] C. Cicconetti, M. Conti, A. Passarella, "A decentralized framework for serverless edge computing in the Internet of Things", *IEEE Trans. Netw. Serv. Manag.*, 2021.

[119] S. Ristov, S. Pedratscher, T. Fahringer, "AFCL: An Abstract Function Choreography Language for serverless workflow specification", *Future Generation Computer Systems*, 2021.

[120] M. S. Aslanpour, A. N. Toosi, M. A. Cheema, R. Gaire, "Energy- aware resource scheduling for serverless edge computing", *Proc. of CCGrid '22*, 2022.

[121] G. Russo Russo, "Serverledge", [Online]. Available:https://github.com/grussorusso/serverledge

[122] Cloud Native Computing foundation, "Wasmedge", [Online]. Available:https://wasmedge.org

[123] The Unikraft Authors, "Unikraft", [Online]. Available:https://unikraft.org/

[124] G. Russo Russo, V. Cardellini, F. Lo Presti, T. Mannucci, "Serverledge: Decentralized Function-as-a-Service for the edge-cloud continuum", *Proc. of 21st Int'l Conf. on Pervasive Computing and Communications (PerCom 2023)*, pp. 131-140, 2023.

[125] The Linux Foundation, "Prometheus", [Online]. Available:https://prometheus.io

[126] J. Arnowitz, M. Arent and N. Berger, "Effective prototyping for software makers", 2010

[127] M. Carr and j. Verner, "Prototyping and software development approaches", *Department of Information Systems*, City University of Hong Kong, pp. 319-338, 1997

[128] L. Conforti, C. Puliafito, A. Virdis and E. Mingozzi, "Server-side QUIC connection migration to support microservice deployment at the edge," *Pervasive and Mobile Computing*, 2022

[129] Divexplorer project, "Divexplorer", [Online]. Available:https://github.com/elianap/divexplorer/tree/main/notebooks

[130] T. Klyver et al., "Jupyter Notebooks-a publishing format for reproducible computational workflows", In *Positioning and Power in Academic Publishing: Players, Agents and Agendas,* pp. 87-90, 2016

[131] A.B. Yoo, M. A. Jiette, M. Grondona, "Slurm: Simple linux utility for resource management*", In Job Scheduling Strategies for Parallel Processing: 9th International Workshop*, pp. 44-60, 2003

[132] Jupyter consortium, "Jupyter Xpra", [Online]. Available:https://github.com/jupyterhub/batchspawner

[133] CINECA, "ADA cloud infrastructure user guide", [Online]. Available:https://wiki.u-gov.it/confluence/display/SCAIUS/UG3.5%3A+ADA+Cloud+UserGuide

[134] CINECA, "Galileo 100 cluster user guide", [Online]. Available:https://wiki.u-gov.it/confluence/display/SCAIUS/UG3.3%3A+GALILEO100+UserGuide

[13 B. Fryxell et al., "FLASH: An Adaptive Mesh Hydrodynamics Code for Modeling

5] Astrophysical Thermonuclear Flashes", *Astroph. J. Supp.* vol 131, no 273, 2000

[136] R.W. Hockney and J.W. Eastwood, *Computer simulations using particles,* Hilger, 1988

[137] Q. Kang et al., "Improving All-to-Many Personalized Communication in Two-Phase I/O", in *SC20 Proceedings of the 2020 ACM/IEEE Conference on Supercomputing*, 2021

[138] Latham et al , "A case study for scientific I/O: improving the FLASH astrophysics code", *Comput. Sci. Discov.* no 5, 2012

[139] C. Ritter et al., "SYGMA: Stellar yields for galactic modeling applications", *Astroph. J. Suppl*. vol 237, no 42, 2018

[140] Flash Center Code Group, "Flash code", [Online]. Available:https://flash.rochester.edu/site/index.shtml

[141] R. Y. Cavana, B. C. Dangerfield, O. V. Pavlov, M. J. Radzicki, and I. D. Wheat, "Feedback Economics : Economic Modeling with System Dynamics", *System dynamics Review*, 2021.

[142] D. L Meadows, W. W. Behrens, D. H. Meadows, R. F. Naill, J. Randers, and E. Zahn. *Dynamics of growth in a finite world*. Wright-Allen Press Cambridge, 1974

[143] D. H.. Meadows, J. Randers, and D. L. Meadows. *Limits to Growth : The 30-Year Update*. Chelsea Green Publishing, 2004

[144] M. J. Du Plessix. *Analyse du modèle World3: sensibilité, dynamique, et pistes d'évolution*. HAL, INSA Lyon, 2019.

[145] H. F. Drake, R. L. Rivest, A. Edelman and J. Deutch, "A simple model for assessing climate control trade-offs and responding to unanticipated climate outcomes", *Environmental Research Letters*, vol 16, no 10, 2021

[146] A. Hay, "Jay Forrester's World2 from 1971 Recreated in C++", [Online]. Available:https://github.com/anthay/World2/, 2021

[147] A. Mignan, "World2 model, from DYNAMO to R", [Online]. Available:https://towardsdatascience.com/world2-model-from-dynamo-to-r-2e44fdbd0975, 2020

[148] , F. C. Moore, J. Rising, N. Lollo, C. Springer, V. Vasquez, A. Dolginow, C. Hope, D. Anthoff. "Mimi-PAGE, an open-source implementation of the PAGE09 integrated assessment model", *Scientific Data*, vol 5, no 1, 2018

[149] C. Vanwynsberghe, "Exploring the Limits to Growth with Python", World Bank Open Data https://towardsdatascience.com/exploring-the-limits-to-growth-with-python-674133874eed, 2021

[150] The World Bank, "World Bank Open Data", [Online]. Available:https://data.worldbank.org/?intcid=ecr_hp_BeltD_en_ext

[151] J. Martensen, C. Rackauckas et al, "DataDrivenDiffEq.jl code", [Online]. Available:https://github.com/SciML/DataDrivenDiffEq.jl

[152] P. Crescenzi, H. Lesfari, E. Natale, A. Rossi, P. B. Serafim "Un framework open-source écrit en Julia pour la modélisation d'évaluation globale intégrée". *ROADEF 2023,* 2023

[153] P. Crescenzi, E. Natale, P. B. Serafim. "WordDynamics.jl code",[Online]. Available:https://github.com/worlddynamics/WorldDynamics.jl

[154] A. Gholami, K. Rao et al, "ROMA: Resource Orchestration for Microservices-based 5G Applications", In *Proceedings of the NOMS 2022-2022 IEEE/IFIP*

Network Operations and Management Symposium, 662, pp. 1–9, 2022

[155] P. Pereira, C. Melo et al, “Availability model for edge-fog-cloud continuum: an evaluation of an end-to-end infrastructure of intelligent traffic management service”, *The Journal of Supercomputing*, vol 78, no 665, pp 4421–4448, 2021

[156] H. Song, R. Dautov et al, “Model-based fleet deployment in the IoT–edge–cloud continuum”, *Software and Systems Modeling*, vol 21, pp. 1931 – 1956, 2022

[157] W. Cerroni, L. Foschini, G. Y. Grabarnik, F. Poltronieri, L. Shwartz, C. Stefanelli, M. Tortonesi, "BDMaaS+: Business-Driven and Simulation-Based Optimization of IT Services in the Hybrid Cloud," in *IEEE Transactions on Network and Service Management*, vol. 19, no. 1, pp. 322-337, 2022

[158] G. Nardini and G. Stea, “Using network simulators as digital twins of 5G/B5G mobile networks,” in *IEEE 23rd International Symposium on a World of Wireless, Mobile and Multimedia Networks (WoWMoM),* pp. 584–589, 2022

[159] D. Van Huynh, V.-D. Nguyen, V. Sharma, O. A. Dobre, and T. Q. Duong, “Digital twin empowered ultra-reliable and low-latency communications-based edge networks in industrial iot environment,” in *ICC - IEEE International Conference on Communications*, pp. 5651–5656, 2022

[160] M. Aldinucci, M. Danelutto, P. Kilpatrick, and M. Torquati, “FastFlow: high-level and efficient streaming on multi-core,” in *Programming Multi-core and Many-core Computing Systems*, 2014

[161] H. Kuchen and C. Murray, “The integration of task and data parallel skeletons*”, Parallel Processing Letters*, vol 12, no 2, pp. 141-155, 2002

[162] C. Murray, “Bringing skeletons out of the closet: a pragmatic manifesto for skeletal parallel programming”, *Parallel computing*, vol 30, no 3, pp. 389-406, 2004

[163] D. De Sensi, T. De Matteis, M. Torquati, G. Mencagli and M. Danelutto. “Bringing Parallel Patterns out of the Corner: the P3ARSEC Benchmark Suite”. *ACM Transactions on Architecture and Code Optimization (TACO)*, vol 14, no 4, 2017

[164] N. Tonci, M. Torquati, G. Mencagli, M. Danelutto. “Distributed-memory FastFlow Building Blocks”, *International Journal of Parallel Programming (IJPP),* Special Issue, 2023,

[165] M. Danelutto, G. Mencagli, A. Ottimo, F. Iannone, P. Palazzari, “FastFlow targeting FPGAs”, in *31st Euromicro International Conference on Parallel, Distributed and Network-Based Processing (PDP)*, 2023

[166] S. Campa, M. Danelutto, M. Goli, H. Gonzalez-Vélez et al., “Parallel patterns for heterogeneous CPU/GPU architectures: Structured parallelism from cluster to cloud”, *Future Generation Computing Systems*, vol 37, pp. 354-366, 2014

[167] G. De Lucia, M. Lapegna, and D. Romano, “A GPU Accelerated Hyperspectral 3D Convolutional Neural Network Classification at the Edge with Principal Component Analysis Preprocessing”, In *Parallel Processing and Applied Mathematics*, Lecture Notes In computer Science vol 13827, pp. 127-138, 2023

[168] G. De Lucia, M. Lapegna, and D. Romano, “Unlocking the potential of edge computing for hyperspectral image classification: An efficient low-energy strategy”, *Future Generation Computer Systems*, accepted, in press, 2023

[169] S. Ernsting, H. Kuchen, “Algorithmic skeletons for multi-core, multi-GPU systems and clusters”, *Int. J. High Perform. Comput. Network*, vol 7, no 2, pp.

129–138, 2012

[170] K. Emoto, K. Matsuzaki, “An automatic fusion mechanism for variable-length list skeletons in SkeTo”, *Int. J. Parallel Program.,* vol 42, no 4, pp. 546–563, 2014

[171] N. Javed, F. Loulergue, “A formal programming model of Orleans skeleton library”, In *PaCT’11*, pp. 40–52, 2011

[172] A. Ernstsson, L. Li and C. Kessler, “SkePU 2: Flexible and Type-Safe Skeleton Programming for Heterogeneous Parallel Systems”, *Int J Parallel Prog* 46, 62–80, 2018

[173] M. Cole, “Bringing skeletons out of the closet: a pragmatic manifesto for skeletal parallel programming”, *Parallel Computing, vol* 30, no3, pp. 389–406, 2004

[174] T. Mattson, B. Sanders, B. Massingill, “Patterns for Parallel Programming”, Addison-Wesley Professional, 2004

[175] Reinders, J.: Intel Threading Building Blocks, 1st edn. O’Reilly & Associates, Inc., Sebastopol (2007)

[176] C. Campbell, R. Johnson, A. Miller, S. Toub, ”Parallel Programming with Microsoft.NET: Design Patterns for Decomposition and Coordination on Multicore Architectures”, Microsoft Press, 2010

[177] E. Pastor, L. De Alfaro, and E. Baralis. "Looking for trouble: Analyzing classifier behavior via pattern divergence." *Proceedings of the 2021 International Conference on Management of Data,* 2021.

[178] E. Pastor et al. "How divergent is your data?." *Proceedings of the VLDB Endowment, vol* 14, no 12, pp. 2835-2838, 2021

[179] S. Sagadeeva, M. Boehm. "Sliceline: Fast, linear-algebra-based slice finding for ml model debugging.", *Proceedings of the 2021 International Conference on Management of Data*, 2021

[180] Y. Chung et al. "Slice finder: Automated data slicing for model validation." *IEEE 35th International Conference on Data Engineering (ICDE)*, 2019

[181] TensorFlow Model Analysis, “Introducing TensorFlow Model Analysis: Scaleable, Sliced, and Full-Pass Metrics” [Online]. Available:https://medium.com/tensorflow/introducing-tensorflow-model-analysis-scaleable-sliced-and-full-passmetrics-5cde7baf0b7b

[182] D. Baylor, E. Breck et al, “TFX: A TensorFlow-Based Production-Scale Machine Learning Platform”, In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining,* 2017

[183] R. K. E. Bellamy, .l Dey, M. Hind et al., “AI Fairness 360: An extensible toolkit for detecting and mitigating algorithmic bias”, *IBM Journal of Research and Development*, vol 63, pp. 4:1–4:15, 2019

[184] P. Saleiro, B. Kuester, A. Stevens et al, “Aequitas: A Bias and Fairness Audit Toolkit”, *arXiv preprint arXiv:1811.05577,* 2018

[185] A. A. Cabrera, W. Epperson et al., “FairVis: Visual analytics for discovering intersectional bias in machine learning”, In *IEEE Conference on Visual Analytics Science and Technology (VAST)*, pp. 46–56, 2019

[186] M. Di Pierro, “Py4Web”, [Online]. Available:https://py4web.com

[187] M. Hussain, L.-F. Wei, A. Lakhan, S. Wali, S. Ali, and A. Hussain, “Energy and performance-efficient task scheduling in heterogeneous virtualized cloud

computing," *Sustainable Computing: Informatics and Systems*, vol. 30, p. 100-517, 2021.

[188] S. S. Roy, F. Turan, K. Jarvinen, F. Vercauteren, and I. Verbauwhede, "FPGA-based high-performance parallel architecture for homomorphic computing on encrypted data," in *2019 IEEE International symposium on high performance computer architecture (HPCA)*, 2019, pp. 387–398

[189] A. V Aho, M. S. Lam, R. Sethi, and J. D. Ullman, *Compilers: principles, techniques and tools*. 2020.

[190] C. Lattner *et al.*, "MLIR: Scaling compiler infrastructure for domain specific computation," in *2021 IEEE/ACM International Symposium on Code Generation and Optimization (CGO)*, pp. 2–14. 2021

[191] A. Bik, P. Koanantakool, T. Shpeisman, N. Vasilache, B. Zheng, and F. Kjolstad, "Compiler support for sparse tensor computations in MLIR," *ACM Transactions on Architecture and Code Optimization (TACO)*, vol. 19, no. 4, pp. 1–25, 2022

[192] W. S. Moses, L. Chelini, R. Zhao, and O. Zinenko, "Polygeist: Raising C to polyhedral MLIR," in *2021 30th International Conference on Parallel Architectures and Compilation Techniques (PACT)*, pp. 45–59, 2021

[193] H.-I. C. Liu, M. Brehler, M. Ravishankar, N. Vasilache, B. Vanik, and S. Laurenzo, "TinyIREE: An ML Execution Environment for Embedded Systems from Compilation to Deployment," *arXiv preprint arXiv:2205.14479*, 2022

[194] LLVM community ,"MLIR dialects", [Online]. Available:https://discourse.llvm.org/t/rfc-updated-mlir-dialect-overview-diagram/64266

[195] "MLIR abstraction for a RISC-V architecture", [Online]. Available:https://github.com/pulp-platform/snitch

[196] LLVM community, "LLVM compiler", [Online]. Available:https://github.com/llvm

[197] Kubernetes community, "Kubernetes Cluster Federation", [Online]. Available:https://github.com/kubernetes-retired/kubefed

[198] The Linux foundation, "Karmada: Open, Multi-Cloud, Multi-Cluster Kubernetes Orchestration", [Online]. Available:https://karmada.io/

[199] The Linux foundation , "Submariner", [Online]. Available:https://submariner.io/

[200] Isovalent Inc, "Cilium Mesh", [Online]. Available:https://isovalent.com/blog/post/introducing-cilium-mesh/

[201] SUSE Rancher, "Fleet", [Online]. Available:https://fleet.rancher.io/

[202] The Linux foundation , "KubeEdge", [Online]. Available:https://kubeedge.io/

[203] OpenInfra foundation, "OpenStack", [Online]. Available:https://www.openstack.org/

[204] Politecnico di Torino, "Liqo running dashboard example", [Online]. Available:https://liqo-dashboard.crownlabs.polito.it/

[205] Politecnico di Torino , "Liqo blog post", [Online]. Available:https://medium.com/the-liqo-blog/liqo-in-production-at-turin-polytechnic-20ed71dca475

[20 Amsterdam Internet Exchange, "Gaia-X test environment", [Online].

6] Available:https://www.ams-ix.net/ams/news/dutch-gaia-x-hub-builds-gaia-x-test-environment

[207] Politecnico di Torino, “Liqo project”, [Online]. Available:https://github.com/liqotech/liqo

[208] Politecnico di Torino, “Liqo documentation”, [Online]. Available:https://docs.liqo.io

[209] M. Iorio, F. Risso, A. Palesandro, L. Camiciotti and A. Manzalini, "Computing Without Borders: The Way Towards Liquid Computing," in *IEEE Transactions on Cloud Computing*, 2022

[210] Politecnico di Torino, “The Liqo Blog”, [Online]. Available:https://medium.com/the-liqo-blog