MISSIONE 4
ISTRUZIONE
RICERCA

SPOKE 1
FUTURE HPC & BIG DATA

FLAGSHIP 3:
Survey of State-of-the-art Approaches and Gap Analysis on HPC dev tools

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 document aims at presenting the state-of-the-art approaches and gap analysis for HPC development tools. Flagship 3 is characterized by a large number of partners with diverse and heterogeneous competencies and a vast amount of HPC tools, ranging from hardware accelerators and runtime optimization tools to resource managers for Edge, Fog and Cloud and different application models. The ultimate goal of this document is to identify the most promising areas for future research and development in HPC development tools. By analyzing the current state-of-the-art approaches and identifying gaps, we hope to guide the Flagship 3 partners towards more effective collaboration and innovation in this field. An approach illustrating the state of the art of each single HPC tool and partner competence would quickly lead to an explosion of technical details. Hence, in this deliverable, we opt for a different approach and focus on a structured presentation of the different competencies and available HPC tools provided by each partner. This approach allows for a clear understanding of the capabilities of each partner and their respective HPC tools, which will aid in selecting the most suitable partner and tool for a specific project or task. Additionally, it provides a comprehensive overview of the overall HPC landscape within the partnership.

In detail, this approach allows us to first learn about the different areas of expertise and HPC tools of the different partners participating in the flagship. Secondly, we organize the available tools and competencies into a structured Hybrid Cloud-HPC architecture to address the development of HPC applications. This architecture spans multiple technological stacks and target platforms. More precisely, this architecture is composed of six layers: the application layer, which provides the functionalities that an HPC application developer needs; the static optimization and transformation layer, which provides the capabilities to translate the high-level application model into code; the virtualization/containerization layer, which targets the execution platform for HPC applications such as Clouds, Edge, and novel HPC resources; the orchestration layer, to manage the lifecycle of applications deployed on the execution platform; the runtime management layer, with specific solutions for the different stages of the application lifecycle; and the hardware layer. The functionalities provided by each layer are detailed in Section 1, and Section 2 presents the tools illustrated in this document, classified within each layer of the proposed Hybrid Cloud-HPC architecture. The presented tools are twenty-seven; eight of them fall within the scope of the application layer, nine within the static optimization and transformation layer, five within the orchestration layer, and four within the runtime management layer. Finally, one tool falls within the scope of the hardware layer. The current state of the art for each tool is presented in Section 3. As with the HPC tools, the HPC applications we will deal with in this Flagship are characterized by very different goals, constraints, and performance metrics. Hence, the available tools require a detailed GAP analysis to identify their current limitations, both from a technological and integration perspective. Section 4 presents an in-depth GAP analysis for the adoption of each tool in the HPC context. Finally, in Section 5, the synergies between possible use cases for HPC applications and some of the proposed tools are briefly introduced.

The HPC architecture defined in this deliverable will be fully exploited in the second 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, assessed to demonstrate the advances achieved with respect to the state of the art at the beginning of this project.

To conclude, this deliverable aims at showcasing the competencies and tools of the partners towards the industrial partners of the Flagship and the Spoke, to allow us an easier deployment of such tools as well as their integration with their specific use cases and to address their specific HPC problems. The introduction of the synergies between possible use cases for HPC applications and proposed tools in Section 5 is crucial to enabling industrial partners to understand the potential benefits of these tools. The aim of this deliverable is to facilitate the deployment and integration of these tools with specific use cases and address their HPC problems.

1 Hybrid Cloud-High Performance Computing architecture

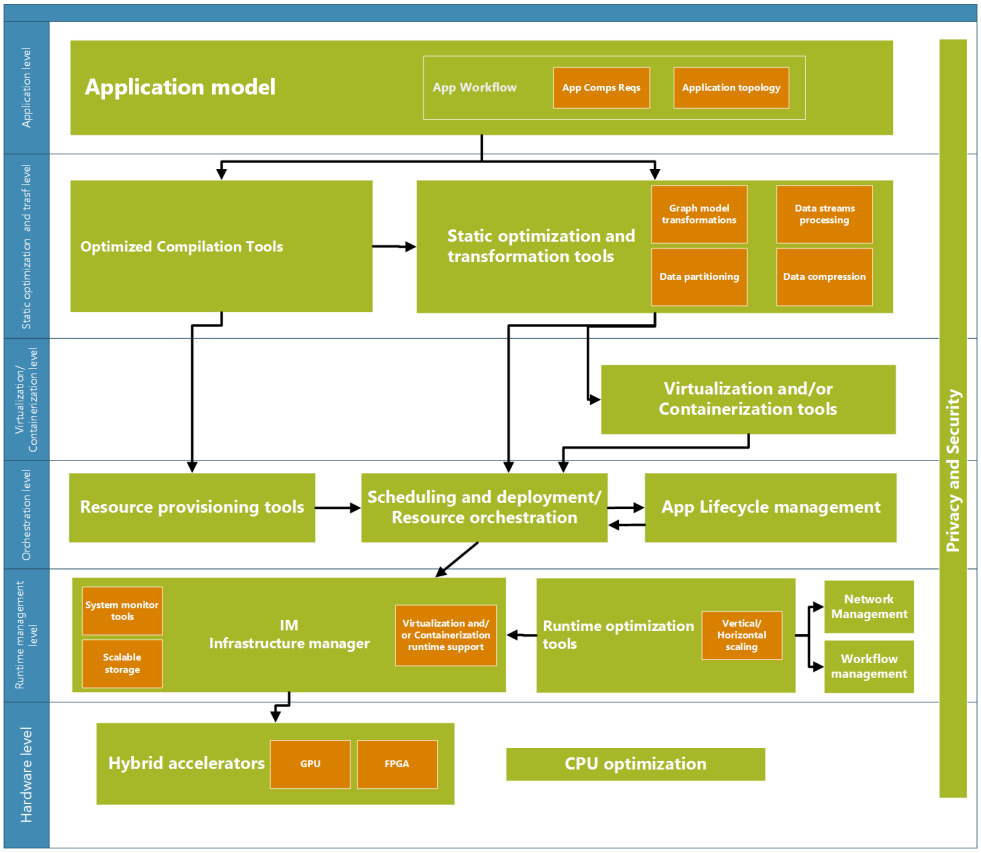

Figure 1 High Performance Computing (HPC) Layers

As the technical barriers to Cloud-based infrastructures lowered substantially with the advent of the *-as-a-Service model, most High Performance Computing (HPC) facilities worldwide are instead still anchored to SSH-based and queue-based job submission mechanisms. Finding an effective way to improve accessibility to this class of computing resources is still an open problem in computer science. Indeed, the multiple layers of virtualization that characterize modern cloud architectures introduce significant processing overheads and make it impossible to apply adaptive fine-tuning techniques based upon the underlying hardware technologies, making them incompatible with performance critical HPC applications. On the other hand, HPC datacenters are not designed for general purpose applications. Only scalable and computationally demanding programs can effectively benefit from the massive amount of processing elements and the low-latency network interconnections that characterize HPC facilities, justifying the high development cost of HPC-enabled applications. Moreover, some seemingly trivial applications are not supported in HPC environments, e.g., exposing a public web interface for data visualization in an air-gapped worker node.

The latest cloud management platforms make it possible to take a hybrid cloud approach, called Hybrid Cloud-HPC, which blends on-premises infrastructure with public cloud services so that workloads can flow seamlessly across all available resources. This enables greater flexibility in deploying HPC systems and how quickly they can scale up, along with the opportunity to optimize total cost of ownership. Such Hybrid Cloud-HPC architecture, depicted in Figure 1, can bring a certain number of benefits:

- **Scalability**: It is easy to serve increasing demands and requirements as it does not require a huge investment and effort. Scaling on-premises HPC can be more expensive and will require additional space to install. Besides, the investment in hardware will be a waste if the requirement is short-term. Instead, it is very convenient to scale cloud requirements on demand while keeping the on-premises resources unchanged.

- **Accessibility**: Another major disadvantage of on-premises HPC is that users should be on-premises to access the resources. On the contrary, Hybrid Cloud-HPC allows users to connect to the running application and to the system to manage, monitor, launch, deploy and collect results through, for example, ad-hoc Internet REST interface or portals.
- **Efficiency**: Moving into the Cloud will help you in many ways. For instance, it will save space, costs, and power consumption, and increase flexibility in the management of the resources, avoiding underutilization of them.
- **Better security**: Cloud-based HPC is an IaaS model. Therefore, it should also offer security, by providing a physical separation between on-premises data and the Cloud, and different applications by separating spaces exploiting virtualization techniques.

In next paragraphs, we explain and describe the functionalities of the various blocks and levels of a Hybrid Cloud-HPC architecture, from the high-level application description and modeling to the low-level hardware optimizations.

1.1 Application layer

This is the layer that provides the functionalities that the application developer needs, i.e., to build or push the application components into the stack and to define their functional or non-functional requirements in terms of needed resources. The application owner must also provide an application model, a description of the application components, their requirements in terms of resources and their expected behavior in case of application- or resource-related events. The most common high-level formalism to represent an application and its requirements and components is a **graph** (usually a DAG, Directed Acyclic Graph). Edges and nodes are annotated with resource requirements, such as latency or bandwidth for edges representing communication channels, or computation cores, quantity of RAM, storage and so on for nodes. This type of formalism have been also adopted in HPC, to model applications so it is possible to abstract from the underlying network topology and to allow the exploitation of Hybrid HPC resources: for example, it is possible to deploy two or more application's components on the same many-cores shared memory machine, or on different machines or, furthermore, on different Clouds, on the basis of the requirements specified on the edges of the graph; in the former case, the components can be implemented as threads and edges as shared-memory channels, while in the latter case the components are processes on different machines, and edges are network channels: the effective deploy scenario is chosen dynamically at the orchestration level, on the basis of the application requirements, to optimize costs and resource consumption. The most common implementation of such formalism are **workflows**. Orchestrating distributed workflows in the compute continuum is a complex task: as could be noted in the example above, execution locations can be heterogeneous, exposing different methods and protocols for authentication, communication, resource allocation and job execution. Plus, they can be independent of each other, meaning that direct communications and data transfers among them may not be allowed. Encoding workflow dependencies and execution environments' topology in a single model allows for optimized execution strategies at different levels of the orchestration stack, from model optimization through consistency-preserving rewriting rules to overhead minimization through data-aware scheduling policies.

It is possible to classify emulators/simulators tools at this level, as a method for fast prototyping and testing applications and to analyze their performance, resource consumption and adherence to functional and non-functional requirements in an environment with reduced costs and controlled behaviors.

Finally, most HPC facilities worldwide are still anchored to SSH-based and queue-based job submission mechanisms: interfaces to simplify interactive sessions to launch and monitor running applications on a Hybrid Cloud-HPC cluster could be classified at this level.

1.2 Static optimization and transformation layer

This is the layer that provides the functionality to translate the high-level application model into code, by performing static transformations and optimizations to increase parallelism, to reduce the impact of communications and general overhead with the objective to increase performance of applications, or to reduce the quantity of needed resources, and consequently costs. Tools of this layer form a software development chain composed, since decades, mainly by high performance optimization compilers, which performs transformations and optimizations based on the high-level application description model specified at the application level described in Section 1.1. In the last decades, such compilers are supported by a set of tools dedicated to other critical optimizations: in Section 1.2.1 the techniques to partition data between nodes are briefly introduced, in particular for data-intensive applications, to increase parallelism and reduce the overhead connected to transferring data during execution; in Section 1.2.2 data streams processing, an approach to process data in real-time by applying a pipeline of operators to them is introduced; in Section 1.2.3 techniques to compress data are introduced; in Section 1.2.4 graph transformations techniques to optimize the graph induced by the application model without changing the semantics of the application are introduced, they increase parallelism or reduce communication overhead.

1.2.1 Partitioning of data for data parallelism applications

Partitioning of data is a technique used in data parallelism applications to distribute large datasets across multiple processing nodes or machines. In data parallelism, large datasets are split into smaller, more manageable chunks and processed simultaneously on multiple processing units. This technique allows for faster processing and analysis of large datasets, making it an essential tool for many modern data-intensive applications. It is a critical component of data parallelism applications as it determines how the dataset will be divided and distributed across the processing nodes, balancing the load to increase parallelism and scalability. There are several methods of partitioning data, such as range partitioning, where data is partitioned based on specific ranges, hash partitioning, which involves hashing the data and round-robin partitioning, the simpler method where data is subdivided circularly between nodes, among others. Each method has its advantages and disadvantages and may be more suitable for specific use cases. The choice of partitioning method depends on several factors, including the nature of the data, the processing requirements, and the available processing resources. Additionally, partitioning of data allows for fault tolerance, as data can be replicated across multiple processing nodes, ensuring that data is not lost in the event of a failure.

1.2.2 Data streams processing

Data streams processing is a modern approach to data processing that involves the continuous and real-time analysis of data as it flows in from various sources. It has become increasingly popular in recent years due to the rise of big data and the need for organizations to process and analyze data in real-time. With the explosion of IoT devices, social media, and other digital technologies, there is an enormous amount of data being generated every second. Batch processing systems suffer from latency problems due to the need to collect input data into batches before it can be processed. Under several application scenarios such as fraud detection in financial transactions and healthcare analytics involving digital sensors and Internet of Things, continuous data streams must be processed under very short delays. This is because certain types of data streams such as stock values, credit card transactions, traffic conditions, time-sensitive patient data, and trending news rapidly depreciate if not processed instantly. Thus, the ability to handle and process continuous streams of data is becoming an essential part of building today’s data-driven organizations, enabling them to

analyze this data quickly and efficiently in real-time, extracting valuable insights, allowing them to make more informed and timely decisions.
This is achieved through the use of distributed and high-parallel frameworks, such as Apache Kafka [1], Apache Flink [2], and Apache Storm [3], which allow data to be processed in parallel across multiple nodes. These systems also provide fault tolerance, ensuring that data is not lost in the event of a failure. Most Modern Data Stream Processing Systems try to combine batch and stream processing capabilities into a single or multiple parallel data processing pipelines.

1.2.3 Compression of data

Data compression is concerned with saving space in data representation. This saving impacts not only on data storage, but also on data transmission and data processing. Another rich dividend that spurs from data compression is energy saving because computing over compressed data uses fewer servers and hardware devices to store, transmit and access data. A drawback of compression comes with the CPU and memory overhead to compress and decompress the data, reducing their applicability in real-cases scenarios such as data streams applications. Thus, the effectiveness of applying data compression is determined by the achievable compression ratio of the selected compression algorithm and the time to compress the data. The future exascale systems are expected to exhibit a trend that the data movement among memory hierarchies and off-node data transfer will become relatively expensive in terms of time and energy, compared to the ever-increasing compute power. If the selected compression algorithm can produce a reasonable compression ratio, the benefit of applying data compression shall become more significant for the scientific applications running on those systems. When it comes to compression, scientists often face the dilemma of choosing **lossless** compression and **lossy** compression. The former preserves data fidelity but can be slow and produce a poor compression ratio, whereas the latter introduces a cumbersome validation process on the approximated data; lossy compressions often produce better compression ratios than the lossless counterpart, but the error rates are not easy to bound, and, in large scale simulations, unbounded error could introduce significant deviation from the actual values.

1.2.4 Graph transformations

HPC has become an essential tool for solving complex scientific and engineering problems that require massive amounts of computational power. As explained in Section 1.1, graphs are a common data structure used to represent applications topology, requirements, and components, usually implemented as workflows. HPC graph transformations are a set of techniques used to optimize graph-based algorithms for HPC systems by transforming the graph to reduce communication and increase parallelism. Processing graphs representing HPC applications can be challenging, as they require massive amounts of computational resources and can generate huge amounts of data. These techniques involve breaking down the graph into smaller subgraphs and applying a series of transformations to each subgraph. Such transformations can include reordering nodes, removing redundant edges, and collapsing nodes into clusters, usually performing what, as an instance, in data streams applications are called the fission or fusion of operators [4]. One of the primary benefits of these techniques is that they can significantly improve the performance of graph-based algorithms on HPC systems, improving their scalability and efficiency, allowing for faster and more accurate results, also reducing their memory and, consequently, energy footprint.

A challenge associated with HPC graph transformations is the need to balance computation and communication. In some cases, optimizing the graph for parallelism can lead to increased communication overhead, which can negatively impact performance. To address this issue,

HPC Graph Transformations must balance the need for parallelism, performing the fission of operators, with the need to minimize communication, performing the fusion of operators, preserving the semantics defined by the operators' ordering and internal operations performed.

1.3 Virtualization/containerization layer

This is the layer that provides functionalities and tools to virtualize/containerize applications. In Cloud infrastructure, virtualization is important to isolate applications, optimize the utilization of hardware resources and provide operational flexibility. However, conventional virtualization comes at the cost of resource overhead, which could be unfeasible for performance critical HPC applications, violating their requirements. Lightweight virtualization could be an alternative, as it potentially reduces overhead and thus improves the utilization of datacenters, allowing for such applications to reach a more fine-grained level of parallelism. A lightweight virtualization method which starts to be employed in Edge/Cloud continuum is containerization: Linux Containers (LXC) [5] is a Linux kernel technology that is able to run a multitude of processes, each in their own isolated environment, sharing the kernel and all the operating system libraries, avoiding to emulate the underlying hardware, while maintaining the applications in isolated spaces; Docker [6] is a widely used tool that makes it easy to package an application and all of its dependencies into such containers.

1.4 Orchestration layer

This is the layer that provides functionalities and tools to orchestrate and manage resources and the lifecycle of applications deployed on a Hybrid Cloud-HPC, which is composed of a "continuum" of resources belonging to public or private clouds, edge clouds and HPC datacenters. This continuum enables the convergence of heterogeneous infrastructures, stretching all the way from cloud to edge devices. The role of orchestration is to manage all heterogeneous types of resources of the Hybrid Cloud-HPC environment, to allow their allocation to the applications running on it by performing an efficient matchmaking of available resources based on the application requirements, with the help of the workload analyzer called *broker*. The broker performs such matchmaking after submission of workloads by user and determines its possibility (whether workload can be provisioned on resources based on QoS requirements or not). Broker sends requests to resource schedulers for scheduling after successful provisioning of resources: tools which try to extend HPC scheduling techniques, such as distributed job scheduling, could be classified at this layer as part of resource orchestration.

Traditional orchestration techniques, such as exact techniques as Mixed Integer Linear programming, are not suitable for HPC or Hybrid Cloud-HPC environments, since their reaction time is too high for real-time adaptation, introducing an intolerable overhead, in particular for fine grained high parallel applications and data streaming applications; efficient application orchestration techniques are needed to manage and coordinate the execution of applications across a diverse range of resources and environments. To this end, proactive and reactive processes are needed to support the application's runtime and adapt its deployment according to the changes in the application workload and resource availability: in particular, approximated distributed techniques such as genetic algorithms, iterative research algorithms, are adapted to solve the problem in a reasonable amount of time and in a scalable way, also reducing the cost in resources needed to run such algorithms.

In this layer are also classified tools to support application lifecycle management, i.e., instantiating and terminating an application, on-demand or in response to a request by an authorized third-party. Orchestrating such a platform in terms of resource allocation and service placement is critical for assuring efficient network resource utilization, Quality of Experience (QoE) and reliability.

To do an efficient and real-time orchestration, there is the need to have a scalable, efficient and near real time resource provisioning service. Its purpose is to retrieve from the underlying runtime management level (in particular, from the IM) up to date information about resource availability, on-demand from the broker, or at a certain interval of time. In a high dynamic environment such as Hybrid Cloud-HPC, such service is usually performed by distributed solutions based on peer-to-peer (P2P) computing such as Distributed Hash Table (DHT) based overlay networks.

1.5 Runtime management layer

This is the layer that provides functionalities to support at runtime the running applications: deleting of applications, migration, storage functionalities, virtualization (if applicable), vertical and horizontal real time scaling, monitoring, and networking management. The functionality of the IM (Infrastructure Manager) is to manage a (possible virtual) infrastructure allowing administrators to modify its configuration, by monitoring and managing the operation of workloads in a heterogeneous resource environment such as Hybrid Cloud-HPC. Thus, it provides monitoring services for the processes running on the cluster nodes, maintains a configuration database and runtime status and interacts closely with virtualization systems, but also can support different CPU architectures and has small footprints in terms of ram and disk. Monitoring information on the availability of resources is reported to the orchestration level to react to changes in Quality of Service (QoS)/QoE and solve violation of the Service Level Agreement (SLA) of the application, as described at high level by the application model using orchestration specification languages such as Topology and Orchestration Specification for Cloud Applications (TOSCA) [7], the OASIS Topology and Orchestration Specification for Cloud Applications, which focuses on complex dependency management, using workflow description languages to write the deployment and management plans.

1.6 Hardware layer

This is the layer that provides functionality to accelerate and optimize the execution of a single application component code, and it represents the lower level of the architecture. At this level, programmers make extensive use of hardware accelerators for specific function classes, such as FPGA (Field Programmable Gate Array) and GPUs (Graphical processing units).

When looking at how hardware influences computing performance, there are general-purpose processors (GPPs) on one end of the spectrum and application-specific integrated circuits (ASICs) on the other. Processors are highly programmable but often inefficient in terms of power and performance. ASICs implement a dedicated and fixed function and provide the best power and performance characteristics, but any functional change requires a complete (and extremely expensive) re-spinning of the circuits. Fortunately, several architectures exist between these two extremes. Programmable logic devices (PLDs) are one such example, providing the best of both worlds. They are closer to the hardware and can be reprogrammed. The most prominent example of a PLD is a field programmable gate array (FPGA). It consists of look-up tables (LUTs), which are used to implement combinational logic; and flip-flops (FFs), which are used to implement sequential logic, plus other discrete components such as BRAMs (block RAMs), digital signal processing (DSP) slices, processor cores, and various communication cores (for example, Ethernet MAC and PCIe). An advantage of FPGA, although running at a clock frequency that is an order of magnitude lower than CPUs and GPUs (graphics processing units), is that, in some cases, they are able to outperform them. In several classes of applications, especially floating-point-based ones, GPU performance is either slightly better or very close to that of an FPGA. When it comes to power efficiency (performance per watt), however, both CPUs and GPUs significantly lag behind FPGAs. Various kinds of FPGA-based systems are available today. They range from heterogeneous systems targeted at high-performance computing that tightly couple FPGAs with conventional CPUs (for example, Convey Computers), to midrange commercial-off-the-shelf workstations that use PCIe-attached FPGAs, to low-end

embedded systems that integrate embedded processors directly into the FPGA fabric or on the same chip.

On the other hand, over the past few years, the GPU has evolved from a fixed-function special-purpose processor into a full-fledged parallel programmable processor with additional fixed-function special-purpose functionality. One of the benefits of the GPU is its large fraction of resources devoted to computation. Allowing a different execution path for each element requires a substantial amount of control hardware. Instead, today's GPUs support arbitrary control flow per thread but impose a penalty for incoherent branching. GPU vendors have largely adopted this approach. Elements are grouped together into blocks, and blocks are processed in parallel. The GPU is designed for a particular class of applications with the following characteristics:

- *Computational requirements are large*. Real-time rendering requires billions of pixels per second, and each pixel requires hundreds or more operations. GPUs must deliver an enormous amount of compute performance to satisfy the demand of complex real-time applications.

- *Parallelism is substantial*. Fortunately, the graphics pipeline is well suited for parallelism. Operations on vertices and fragments are well matched to fine-grained closely coupled programmable parallel compute units, which in turn are applicable to many other computational domains.

- *Throughput is more important than latency*. GPU implementations of the graphics pipeline prioritize throughput over latency. The human visual system operates on millisecond time scales, while operations within a modern processor take nanoseconds. This six-order-of-magnitude gap means that the latency of any individual operation is unimportant. Consequently, the graphics pipeline is quite deep, perhaps hundreds to thousands of cycles, with thousands of primitives in flight at any given time. The pipeline is also feed-forward, removing the penalty of control hazards, further allowing optimal throughput of primitives through the pipeline. This emphasis on throughput is characteristic of applications in other areas as well.

2 Tools to be adopted in the Hybrid Cloud-HPC architecture

In this Section, the tools developed and/or maintained by the Flagship partners are briefly presented and discussed with respect to each layer of the proposed Hybrid Cloud-HPC architecture. The tool presentations are organized considering the different layers introduced in the description of the Hybrid Cloud-HPC architecture, i.e., Application level, Static Optimization and Transformation level, Orchestration level, Runtime Management level and Hardware level.
All the presented tools have been developed by one or more partners of the Flagship. They build up the core set of HPC tools and competencies that can be leveraged during the forthcoming activities of the Flagship. Each component is described with respect to the reference scenario or problem it addresses.

2.1 Application layer

2.1.1 BDMaaS+

BDMaaS+ [8] is a decision support tool for service providers who want to distribute an IT service on a global scale relying on private and public cloud platforms. To find an optimal IT service configuration, BDMaaS+ relies on a Simulator for IT Service in Federated Clouds (SISFC), a simulator that allows its users to reproduce the behavior of an IT service using realistic latency modeling between different data centers. Both SISFC and BDMaaS+ are available to the interested research community and practitioners on GitHub. The purpose of BDMaaS+ is to find a configuration for an IT service that allows minimizing resource rental costs (Virtual Machines on Cloud platforms) and at the same time providing an adequate level of performance that satisfies Service Level Agreements (SLAs) stipulated by the service providers with their customers. BDMaaS+ implements an optimization component that reenacts an IT service using a specific configuration for evaluating possible alternative service placement configurations over the Hybrid Cloud-HPC environment. This optimization component leverages a two-phase memetic algorithm that mimics -- leveraging the SISFC simulator -- possible service placements among those generated by the modeling stage considering not only the locations of VMs in the Hybrid Cloud-HPC but also the size/flavor of these VMs.

2.1.2 WorldDynamics

WorldDynamics.jl [9] aims to provide a modern framework to investigate integrated assessment models (IAM) of sustainable development, based on current software engineering and scientific machine learning techniques. WorldDynamics.jl is developed in Julia exploiting its libraries to allow scientists to easily use and adapt different world models, from Meadows et al.'s World3 [10] to recent proposals. By enabling an open, interdisciplinary, and consistent comparative approach to scientific model development, the main goal is to inform global policy makers on environmental and economic issues.
In particular, each IAM can be seen as a set of subsystems. Each subsystem is described and implemented by a set of variables and a set of differential-algebraic equations which models their interactions; as of today, WorldDynamics.jl leverages the library ModelingToolkit.jl and its ability to compose and solve them. As such, the goal of each model is to understand the evolution of some major variables. WorldDynamics.jl emphasizes this modular structure of IAMs by facilitating the coding of the systems of equations corresponding to the different subsystems and their composition by automatically deriving the connections among them (that is, their shared variables).
Moreover, WorldDyanmics.jl allows the execution of sensitivity tests as well as the implementation of new models by changing a single subsystem of equations or the whole

IAM providing a brand new set of variables and equations. Those are possible thanks to WorldDynamics.jl's modularity.

2.1.3 Real Time Simulator for Digital Twin and Hardware-In-Loop in the Electrical Power Networks Scenario

Real Time Simulators (RTSs) are gaining momentum in power network monitoring [11], [12]. In fact, RTSs and virtual models of physical devices should be characterized and assessed as well to avoid unexpected sources to the overall uncertainty.
In this scenario, UNIBO performed some research activities, including:

- a calibrator for RTS systems has been designed and characterized;
- an open-Source MATLAB-Based virtual phasor measurement unit (PMU) Library has been developed [13]. This PMU is compliant with the IEC/IEEE 60255-118-1 standard;
- the described calibrator has been used to characterize a PMU developed inside an RTS [14];
- a Stand Alone Merging Unit (SAMU) based on real-time HIL technology has been developed along with its characterization procedure, thus allowing to assess its performance from a metrological perspective [15].

2.1.4 Interactive Computing Service

The infrastructure put in place to provide the InterActive Computing (IAC) service is composed by two components:

- the front-end side featured by a user interface that handles authentication as well as the near-immediate access to the computational resources. The amount of such dedicated resources can be chosen by the user after the login stage via an ad-hoc form. Different authentication methods can be implemented, such as password-based, multi-factor, token, and others;
- the back-end side, where a Jupyterlab server is initialized via a batch job with near-immediate access (thanks to oversubscribe settings). This allows a user-friendly experience thanks to both the "standard" Jupyterlab platform, and the custom plugins added.

The deployment of the IAC framework involves both the front-end and the back-end part. Currently, the development environment is hosted on CINECA infrastructure, with the front-end on the HPC ADAcloud [16] infrastructure and the back-end on GALILEO100 HPC cluster [17].
In the figure below a sketch of this development IAC service is provided, focusing on both the front-end (yellow part) and back-end (light blue part) part.

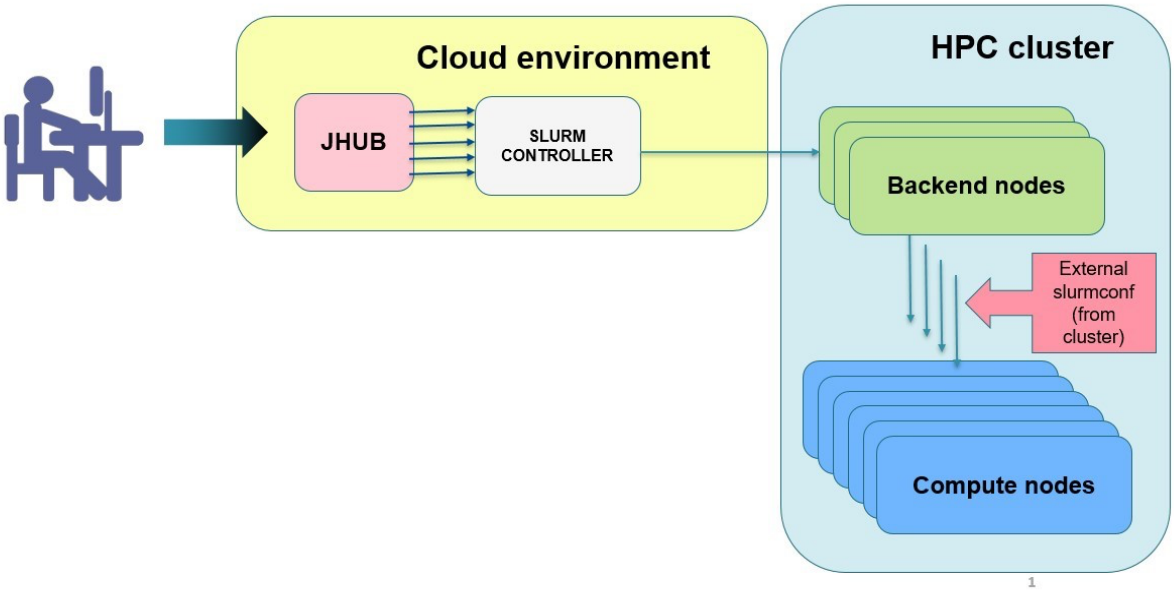

Figure 2 Interactive Computing Server sketch

The development environment depicted allowed us to test new features as soon as they were added, in view of a future production platform.
The browser-based access to the IAC service allows to achieve two different goals:

- the user can access the HPC resources using a much more user-friendly approach with respect to the traditional HPC access tools (mostly ssh). Moreover, the web interface allows to monitor and display results during the execution of a job in an interactive fashion;
- such an interactive approach overcomes the main limitation of the "traditional approach" for HPC resources, which is usually handled via batch job. A batch job, in fact, does not usually allow modifying the execution of the workflow during its execution, and results are managed only at the end of the run; vice versa, an interactive approach allows the user to build the workflow through a real-time interaction, widely enhancing flexibility for an optimal usage of the allocated resources.

2.1.5 Jupyter Workflow

Jupyter Workflow [18] (https://jupyter-workflow.di.unito.it) is an extension of the IPython kernel designed to support distributed literate workflows. The Jupyter Workflow kernel enables Jupyter Notebooks to describe complex workflows and execute them in a distributed fashion on Hybrid Cloud HPC infrastructures. In particular, code cells are regarded as the nodes of a distributed workflow graph. In contrast, cell metadata are used to express data and control dependencies, parallel execution patterns (e.g., Scatter/Gather), and target execution infrastructures (e.g., HPC facilities, Cloud VMs, Kubernetes). Relying on cell metadata to describe workflows has several significant advantages. First, it maintains a clear separation between host and coordination semantics, improving the readability and maintainability of complex applications. Second, it avoids technology lock-in: the same metadata format can be interpreted by different Jupyter kernels to support more languages (other than Python), specific execution architectures, or commercial software stacks. Finally,

it smooths the learning curve of stand-alone workflow systems. Users familiar with Jupyter Notebooks do not have to learn a new framework to scale their experiments.

2.1.6 StreamFlow

The StreamFlow framework [19] (https://streamflow.di.unito.it/) is a container-native Workflow Management System (WMS) written in Python 3 and based on the Common Workflow Language (CWL) open standard [20]. It has been designed around two main principles. First, it allows the execution of tasks in multi-container environments to support the concurrent execution of multiple communicating tasks in a multi-agent ecosystem. Second, it relaxes the requirement of a single shared data space to allow for hybrid workflow executions on top of multi-cloud or Hybrid Cloud HPC infrastructures. StreamFlow declaratively describes cross-application workflows with data dependencies, complex execution environments composed of heterogeneous and independent infrastructures, and mapping steps onto execution locations. This hybrid workflow approach enables the deployment of different, potentially distributed workflow steps (e.g., MPI, TensorFlow) onto different modules (e.g., HPC facilities, Cloud VMs, Kubernetes). StreamFlow allows the seamless integration of new modules and deployment methods through self-contained plugins. The topology awareness emerging from these workflow models allows StreamFlow to implement locality-based scheduling strategies, automated data transfers, and fault tolerance.

2.1.7 Parallel Multi-density Clustering

Recently UNICAL proposed the City Hotspot Detector (CHD), a multi-density based approach to detect urban hotspots in a city [21]. In a nutshell, the algorithm behaves as follows. First, the neighborhood density for each point is computed. Second, on the basis of density variations, the points are partitioned into several density level sets, each one characterized by homogeneous density distributions. During this step, a moving average filtering is performed to smooth out strong density fluctuations (very frequent fluctuations between subsequent values, often occurring in the analysis of real-world urban data) and highlight main trends. Then, each density level set is analyzed by a specific density-based clustering algorithm, to detect clusters in the data partition. The final result of the algorithm consists in a set of spatial clusters, each one representing an urban hotspot. A preliminary experimental evaluation, carried out on state-of-art datasets, have shown good results in terms of clustering accuracy.

2.1.8 aMLLibrary

aMLLibrary (https://github.com/aMLLibrary/aMLLibrary) is a high-level Python package that allows training of multiple performance models, supporting feature selection and hyperparameters tuning. It is based on the scikit-learn toolkit (https://github.com/scikit-learn/scikit-learn) and uses supervised ML techniques to generate regression models which can be used to predict applications performance. Overall, aMLLibrary implements an autoML solution, i.e., it performs training of multiple regression models and automatically selects the most accurate one based on the validation metric chosen. The execution of the library is controlled by a simple configuration text file (or equivalently, a Python dictionary), where the user can specify the dataset to be used, the training settings, the regression models to be tested and their ranges of hyperparameters, and the validation method.
The library currently supports Decision Tree (DT), Non-Negative Least Squares (NNLS), Random Forest (RF), Ridge Linear Regression, Stepwise (a linear regression model which integrates the Draper-Smith feature selection technique, see [22], Support Vector Regression

(SVR), and XGBoost ([23]). Hyperparameter tuning of these models can be performed either via grid search by specifying the lists of values to be tested, or automatically via Bayesian Optimization (BO). If choosing BO, the HyperOpt library (https://github.com/aMLLibrary/hyperopt) is used, with which aMLLibrary is integrated. In this case, the user must provide the appropriate flag in the configuration file, as well as prior probability distributions on the hyperparameters to be optimized by BO. A combination of both tuning methodologies can also be used. The user can choose among several validation methods to compute the Mean Absolute Percentage Error (MAPE) of a model, which is computed as:

$$MAPE(y, y') = \frac{1}{N} \sum_{i=1}^{n} \left| \frac{y_{i-} y'_i}{y_i} \right|$$

where y is the vector of true values and y' is the vector of predicted values by the ML model. The validation methods include classical ones such as train-test splitting and Cross-Validation (CV), and other methods often used in ICT settings to build custom test sets, such as interpolation and extrapolation. Here, interpolation means keeping some feature values within the feature space out of the training set, and placing them in the test set, to check the ability of the model to “fill in the blanks” of the feature space. On the contrary, extrapolation means keeping an entire area of the search space out of the training set, in order to test the predicting capabilities of the model in an unexplored part of the feature space. After the validation phase, the best model is chosen according to the smallest validation MAPE found, and it is saved to file in binary form. This way, it can be used for further inference and to estimate the performance of a given application component for design-time state space exploration, or for runtime resource management.
Finally, the library has a prediction module that can be used to make interpolation and extrapolation with a trained regression model.

2.2 Static optimization and transformation layer

2.2.1 Sieve, Process, and Forward (SPF)

Sieve, Process and Forward (SPF) is a Fog-as-a-Service platform developed in response to the particular needs and challenges posed by Smart City environments. SPF proposes a new information-centric service model based on Value of Information (VoI) methodologies and tools. Value-of-Information (VoI) is a subjective metric that enables ranking information objects to quantify the utility that can bring to their consumers. The SPF VoI-based model allows service providers to define self-adaptive and composable services, which can be deployed as multiple service components that run across the Compute Continuum, migrate to different computing platforms, and automatically scale computational and bandwidth requirements based on the current execution context by filtering the information objects to be processed and disseminated. SPF has two main components: Programmable IoT Gateways (PIG) and a Controller. PIGs provide data processing and information dissemination functions in response to user requests to the platform and can be implemented in the fog (i.e., on edge devices) or in the cloud. One instance of the PIG component must be deployed on each fog or cloud node capable of running an SPF service. The Controller component allows application developers to define and deploy service components on PIGs according to custom deployment policies. Furthermore, the controller handles user requests and forwards them to the relevant PIGs (i.e., those that have running instances of the corresponding Fog services) for processing. Finally, the Controller also implements sensor detection capabilities and can connect PIGs to new data sources. By controlling the service and the implementation

of the application, the Controller ensures that the data processing on the PIGs takes place only when necessary, for example in the presence of user requests.

2.2.2 ParSoDA: Parallel Library for Big Data Analysis

ParSoDA (Parallel Social Data Analytics) [24] is a Java programming library for simplifying the development of parallel data mining applications executed on HPC systems. To achieve this goal, ParSoDA provides a set of widely used functions for processing and analyzing data collected from semi-structured sources. ParSoDA defines a general parallel framework for a data analysis application that includes a number of steps (i.e., data acquisition, filtering, mapping, partitioning, reduction, analysis, and visualization), and provides a predefined (but extensible) set of functions for each data processing step. Thus, an application developed with ParSoDA is expressed by a concise code that specifies the functions invoked at each step. For each of these steps ParSoDA provides a predefined set of functions. Users are free to extend this set with their own functions. For example, for the data acquisition step, ParSoDA provides crawling functions for gathering data from some of the most popular social media (Twitter and Flickr), while for the data filtering step, ParSoDA provides functions for filtering geotagged items based on their position, time of publication, and contained keywords. ParSoDA was capable of supporting the execution of applications on top of both Apache Hadoop and Apache Spark.
To enable further runtime support, a port of ParSoDA to Python is in progress. In such a way, it will be possible to easily extend ParSoDA to support other Python-based runtimes for HPC systems, such as COMPSs [25] by using its Python binding PyCOMPS [26]. PyCOMPSs is a framework, built on top of COMPSs, that facilitates the development of parallel computational workflows in Python. In this approach, users program their scripts in a sequential fashion and decorates the functions to be run as asynchronous parallel tasks. A runtime system is in charge of exploiting the inherent concurrency of the script, detecting the data dependencies between tasks and spawning them to the available resources.
In the ParSoDA-PyCOMPSs version, the support for Distributed Data Set (DDS) will be added. DDS is a lightweight library that provides an interface where programmers can load data from basic Python data structures, generators, or files, distribute the data on available nodes, and run some most common big data operations on it. By using DDS, the number of code lines can be reduced, where performance improvement is not expected compared with regular PyCOMPSs applications.

2.2.3 BLEST-ML (BLocksize ESTimation via Machine Learning)

BLEST-ML (BLocksize ESTimation via Machine Learning) [27] addresses the problem of data partitioning by finding a suitable estimate of data block size, allowing an effective hybrid partitioning of a given dataset to be processed by a data-parallel algorithm in a target distributed environment. This can help programmers to make the most of all the computing and storage resources that are available in the environment, as they can efficiently obtain a suitable estimate for the block size, without the need for heavy tuning processes or domain knowledge.
As a preliminary step, the methodology requires a careful analysis of the execution environment, generally characterized by a set of software features, such as the available frameworks and libraries, and infrastructure features, such as the number of nodes, cores per node, available memory, and disk space. BLEST-ML leverages a log of past executions to extract the patterns that link a specific execution to the best block size, by training a supervised machine learning model. However, in order to learn effective patterns, raw logs must be adequately processed to extract an appropriate set of training data. The log consists of a collection of executions, performed by both standard users and domain experts, in which

a single execution is described by: *(i)* the characteristics of the dataset, *(ii)* the algorithm, *(iii)* the execution environment, *(iv)* the specific partitioning applied along rows and columns, *(v)* the overall execution time, and *(vi)* other measurements such as main memory/disk usage. The methodology also provides a mechanism for log creation and/or enrichment based on a grid search procedure.
Given the dataset extracted from the execution log, a classification model is trained to learn the patterns that relate the execution features/parameters and the best partitioning. Thus, the output of this step is a classification model capable of estimating the optimal number of partitions in which to split the rows and the columns of a given dataset, based on its characteristics, the algorithm to be run, and the underlying execution environment. In order to support hybrid partitioning (i.e., both horizontal and vertical), which implies the prediction of a two-dimensional variable (block rows and columns), a multi-output classification model based on a cascade of two decision tree classifiers is used. In particular, as the two target variables are likely to be dependent on each other, BLEST-ML leverages a chained model to condition the prediction of the block columns on that of rows.

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

UNIPI has a long-time expertise in designing data compressors. In [28] UNIPI addressed the problem of HTML text-file compression via the so-called PPC paradigm: Permuting + Partition + Compress, whose main algorithmic idea is to first permute the files in order 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).
The PPC paradigm allows to compress collections of several billions of texts and source code files (written in markup and programming languages, thus not just HTML) to achieve effective compression ratios and efficient (de)compression speed in two different scenarios: Backup and RandomAccess. The former is concerned with the storage scenario in which only a streaming access to the whole compressed collection is supported; the latter is concerned with the case in which efficient access to individual files of the compressed collection is supported. Several new instantiations of the PPC paradigm are being investigated: 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, also the use of compressed indexes (à la FM-index or CSA) is being investigated in order to achieve entropy-bounds in space occupancy and still preserving the ability to decompress only the requested file, and not much more.

2.2.5 MALAGA, MultidimensionAL Big DAta Analytics over Massive Graph DAta

MALAGA (MultidimensionAL Big DAta Analytics over Massive Graph DAta) is a Hadoop-compliant Java-based open source framework for supporting all the tasks of the multidimensional big data analytics procedure, which can be summarized as follows:

- data connection/alimentation over the massive graph data source, even in multiple fashion;
- ETL (Extraction, Transformation & Loading) over the massive graph data source, even in multiple fashion;
- definition of the multidimensional data model over the graph data source, according to the underlying big data analytics goals;

- data partitioning/distribution of the so-generated big multidimensional data structures over the target commodity Cloud;
- definition of the multidimensional big data analytics tasks over the target big multidimensional data structures, from conventional ones (e.g., drill-down, roll-up, pivoting etc.) to advanced ones (e.g., user-defined aggregations, etc.);
- definition of ad-hoc interactive dashboards over big multidimensional data analytics.

It should be considered that, from a proper research perspective, implementing and realizing MALAGA encompasses several relevant research challenges. Indeed, while some research proposals have investigated the problem of supporting conventional OLAP operations over graph data (e.g., [29]), the problem is still an open research issue, particularly when considered as related to the emerging big graph data context.

2.2.6 FastFlow/WindFlow: high-level and efficient streaming

FastFlow is a parallel programming library (https://github.com/fastflow/fastflow), initially targeting multi/many-core architectures [30], leveraging the principles of structured parallel programming methodology. Recently, it has been extended to target distributed-memory platforms [31]. FastFlow aims to define a *single programming model* for shared- and distributed-memory systems leveraging a streaming data-flow programming approach and a reduced set of structured parallel components called Building Blocks (BBs). *FastFlow's* BBs provide the programmer with efficient and reusable implementations of essential parallel components that can be assembled following a LEGO-style model to build and orchestrate more complex parallel structures (including well-known algorithmic skeletons and parallel patterns). FastFlow's run-time currently targets MPI and TCP/IP backends [31]. Ongoing efforts aim to transparently extend its communication backend to target UCX, MQTT, and RabbitMQ.
WindFlow [32] is a high-level data stream processing library written on top of FastFlow's BBs. It allows streaming applications featuring special-purpose operators (e.g., map, filters, window-based operators) to be quickly developed by the users and connected in arbitrary acyclic topologies to be executed on multicore architectures. Recently, WindFlow has been extended to support hybrid execution on CPU+GPU architectures, with a design compatible with traditional servers and System-on-Chip devices equipped with integrated GPUs.

2.2.7 Clustering Algorithm

A critical constraint of the clustering algorithm is that it can be challenging to define the number of clusters in specific problems from the input data. To overcome this, an algorithm which adjusts such value until it meets the user's desired quality criteria has been developed [33] [34]. The algorithm focuses only on clusters with low affinity to reduce the computational cost primarily associated with element displacement among clusters. The procedure may be terminated when the similarity of the elements within the clusters exhibits minimal or negligible change. Examination of the computational kernel of the algorithm reveals different models of parallelism that can be applied to the data structures managed therein. More precisely it is possible to find a parallelism at cluster level as well as at element level. For such a reason, it is possible to implement this kernel through a hybrid procedure that splits the original dataset into two subsets where the number of elements assigned to each computing unit can be set proportional to its performance ratio.

2.2.8 High performance and Low Power Hyperspectral Image Analysis

In general, for Hyperspectral Imaging (HSI) applications, a reasonable compromise between high performance and low energy consumption is achieved through the combined use of methodologies aimed at modifying the model through mathematical techniques employed to filter out unimportant parts of the model using data compression techniques (such as Principal Component Analysis) and through computational acceleration techniques based on different forms of parallel computing (for example, through specialized libraries such as PyTorch). From a hardware perspective, an interesting tool for assessing the actual usage of the techniques mentioned with reduced energy consumption is the Nvidia Jetson Nano board, which can be connected to various types of remote sensors for edge computing. A prototype with these characteristics is being consolidated at the operating unit of the University of Naples. It is also interesting to note how the adoption of distributed edge computing infrastructures can mitigate some weaknesses of the typical centralized approach of cloud computing, such as constraints on communication network bandwidth and difficulty in implementing real-time applications that use data collected from sites distant from computing infrastructures [35].

2.2.9 DivExplorer: Analyzing Machine Learning Model Behavior via Pattern Divergence

DivExplorer [36], [37], 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 proposed 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.

2.3 Orchestration layer

2.3.1 BookedSlurm

BookedSlurm is a collection of tools extending the traditional Slurm functionalities. The main component is a web calendar, accessed by users to check the current state of the Slurm cluster, showing which computational nodes are currently in use and which resources are reserved for other users in the future and allowing them to reserve resources. Web calendar reservations resemble the concept of urgent computing in a shared environment: users can book resources as soon as they are guaranteed to be free, jumping in front of the job queue before another job from the FIFO has been scheduled. In order to compensate for the time advantage given to users by this approach, the web calendar comes with a credit system: each reservation has a predetermined cost dependent on the resource type, number and the computation time duration. Users have to purchase credits as in a pay-per-use computing system. The second tool is a Slurm job submit plugin, which extends Slurm functionalities by synchronizing the resource state with the web calendar and mapping each job's billing value to the value of its corresponding credit on the web calendar. This tool allows us to keep the

pay-per-use model while giving different costs to jobs executed by submitting to the default Slurm queue or reserving resources through the web calendar. The last tool is a wrapper to provide the interactions between the web calendar and Slurm, offering REST APIs to create and delete reservations.

2.3.2 Orchestration of composite containerized applications in the Cloud continuum

TORCH [38] is a framework for the deployment and orchestration of cloud resources and services in multi-cloud environments. TORCH leverages the TOSCA specification to build up a cloud service orchestrator capable of automating the execution of tasks and operations required for the provisioning of a multi-cloud application. TORCH aims to ease the deployment of cloud applications and to streamline the management of applications lifecycle.
The basic strategy adopted by TORCH is to convert a TOSCA cloud application model into an equivalent BPMN [39] workflow and dataflow model, which a BPMN engine leverages to enforce the operations specified in the model. TORCH main features include: description and modeling of the application topology using standard languages (namely, TOSCA); capability to deploy application components over many cloud providers' platforms, by means of pluggable "connectors"; integration with different container-based cluster technologies; fault-aware orchestration based on a set of business process models; management of deployments is done via a simple web tool.
The INDIGO orchestrator [40] [41] is an open source tool for the deployment and management of composite containerized applications in the Cloud continuum. The tool allows users to set up and build virtualized computing infrastructures and applications with complex topologies (such as clusters of virtual machines or applications packaged as pipes Docker containers), by leveraging standardized interfaces based on REST APIs and adopting the TOSCA templating language. From the received TOSCA-compliant request, the Orchestrator implements a complex provisioning workflow aimed at fulfilling the request using information about the health status and capabilities of underlying virtualized computing infrastructure and their resource availability, QoS/SLA constraints, the status of the data files and storage resources needed by the service/application. The tool interfaces with several virtualized computing frameworks (e.g., Openstack, Apache Mesos [42], Kubernetes) as well as with some of the most relevant public Cloud providers (Amazon and Microsoft Azure).
Assisted by very simple guidelines, application developers can easily set up, configure, deploy and run distributed applications over a continuum of Cloud and Edge resources. The strength of the tool is its capability of abstracting the complexity of orchestrating resources and services in heterogeneous computing environments. Such a level of abstraction is realized by means of lightweight containerization technologies (typically, Docker) and well-recognized standards for the representation of virtualized applications showing complex topologies (the TOSCA standard, indeed).
The tool can remotely control the computing resources offered by an Edge machine, i.e., a computing node with limited capacity. The tool was successfully tested on COTS hardware. To enable such a remote control, the Mesos-Edge software [43] is preliminarily installed on Edge machines. The ability to orchestrate on the Edge both long-term tasks and short-lived jobs is supported by two tools: Marathon [44], a production-grade container orchestration framework that can launch applications and provide scaling and self-healing for containerized workloads, and Chronos [45], a fault-tolerant job scheduler.

2.3.3 Liqo

Liqo is an open-source project that enables dynamic and seamless Kubernetes multi-cluster topologies, supporting heterogeneous on-premises, cloud and edge infrastructures, becoming a fundamental ingredient for the upcoming computing continuum. It provides:

- **Peering**: Automatic peer-to-peer establishment of resource and service consumption relationships between independent and heterogeneous clusters. No need to worry about complex VPN configurations and certification authorities: everything is transparently self-negotiated for you.
- **Offloading**: Seamless workloads offloading to remote clusters, without requiring any modification to Kubernetes or the applications themselves. Multi-cluster is made native and transparent: collapse an entire remote cluster to a virtual node compliant with the 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 CNI plugins. 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. Seamlessly extend standard (e.g., database) high availability deployment techniques to the multi-cluster scenarios, for increased guarantees. All without the complexity of managing multiple independent cluster and application replicas.

Liqo is available at https://liqo.io.

2.3.4 Energy efficient orchestration and resource management in the cloud continuum

UNIPI has recently developed an energy-efficient resource management algorithm to manage the placement of VMs in a cloud environment considering the QoS requirements of the VMs while minimizing the energy footprint of the overall platform [46]. The proposed approach, in addition to managing the initial placement of VMs in one of the servers of the cloud platform based on the current resources available and the resources required by the VM, also includes a dynamic component that is responsible for monitoring the status of the overall cloud infrastructure and applying energy optimization via load redistribution. Such a dynamic component continuously monitors the status of each server, e.g., its load, the network traffic, RAM occupation, etc., and based on the current metrics perform energy optimizations by shutting down some of the servers in the infrastructure, whenever it is possible. To this aim, the proposed component manages the redistribution of VMs from an underutilized server to another server of the infrastructure to shut the server down. On the other hand, if the overall infrastructure is running out of available resources and it is near the overload threshold, some powered-off servers are powered on again to make some additional capacity available. The proposed solution has been demonstrated to be effective via performance evaluation based on realistic data from a cloud provider.

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

As the Function-as-a-Service (FaaS) paradigm experiences a growing popularity within Cloud-based services, there is increasing interest in moving serverless functions towards the Edge, to better support geo-distributed and pervasive applications. However, enjoying both the reduced latency of Edge and the scalability of FaaS requires new decentralized architectures and implementations to cope with typical Edge challenges (e.g., nodes with limited

computational capacity). While first solutions have been proposed for Edge-based FaaS, including light function sandboxing techniques, there is still the lack of a platform with the ability to span both Edge and Cloud and adaptively exploit both [47].
Serverledge [48], 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. Serverledge also accounts for differentiated 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.
The experimental evaluation has shown that Serverledge outperforms Apache OpenWhisk [49] in an Edge-like scenario and has competitive performance with state-of-the-art frameworks optimized for the Edge [50] [51] [52], with the advantage of built-in support for vertical and horizontal offloading.
Serverledge has been designed 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. The code is available at https://github.com/grussorusso/serverledge.

2.4 Runtime management layer

2.4.1 MoveQUIC: a QUIC-based toolbox for the live migration of microservices at the network edge

To support the live service migration of microservices at the edge a toolbox has been developed. The toolbox is based on three main components: CRIU to support container migration, an extended migration-enabled version of the QUIC protocol, an edge platform based on an extension of the ETSI Multi-access Edge Computing (MEC) standard. In the following, the characteristics of the three toolbox components are detailed.
The first component, CRIU, is currently the most used tool to checkpoint and restore the status of containers. CRIU is used to checkpoint in-memory state (e.g., memory pages or variable values at application-level) as a collection of files on disk, and to restore the container from that checkpoint in a later moment. Within the scope of the toolbox CRIU is complemented with a file transfer mechanism, such as rsync, to copy the container state from the source to the destination server. The toolbox supports, through CRIU, four main migration techniques, which differ on the way the checkpoint is created and transferred:
1) Cold migration: it (i) starts by stopping the execution of the container thus ensuring the application state is no longer modified; (ii) then performs a checkpoint of the state and transfers it from the source to the target node while the container execution is still frozen; and (iii) finally resumes the container execution at destination. The cold migration technique typically causes long container downtimes (i.e., the time interval during which the container is not running). As a matter of fact, the downtime in this case equals the overall time required to transfer the container state.
2) Pre-copy migration: it starts with a first phase (i.e., pre-dump), wherein it checkpoints and transfers the container state while the container is still running. Then, in a second phase (dump phase), it stops the container and transfers only the modifications to the state that

occurred during the pre-dump phase. Finally, it restores container execution at destination. In this case, the downtime is typically shorter than the case of cold migration, as only modifications to the state are transferred. However, the data transferred during migration is larger as part of the state (i.e., that being modified) is copied more than once.

3) Post-copy migration: it starts by suspending the execution of the container at the source node and creates a checkpoint containing a minimal part of the state, which includes CPU state and registers content. This checkpoint is copied at the destination, allowing container execution to resume there. Then, while the container is in execution at destination, the rest of the state (i.e., faulted pages), which is most of it, is transferred in the background. During this last phase, container execution may have degraded performance as a considerable part of the state is missing. However, downtime for post-copy migration is typically very short.

4) Hybrid migration: it behaves as a combination of pre-copy and post-copy. The pre-dump phase is the same as for pre-copy migration. The container is then stopped, and the CPU state and registers are copied to the destination. Finally, the container resumes its execution at destination, and, while it is running, the remaining part of the state (i.e., faulted pages) that was modified during the pre-dump phase is copied to the destination. In general, hybrid migration presents the shortest downtime, but shares the same shortcomings with pre-copy and post-copy.

The second component is an extended version of the QUIC protocol. QUIC is a connection-oriented transport protocol, standardized by IETF, that is gaining momentum as a key element in the communication of microservice-based applications. QUIC runs over UDP but provides reliable communication through the implementation of mechanisms such as flow control, congestion control, and loss detection. QUIC outdoes TCP in several aspects. Firstly, TCP runs in kernel space, which means that pushing changes to TCP stacks typically requires operating system upgrades. On the other hand, QUIC is implemented in user space. Secondly, QUIC avoids the head-of-line blocking problem that afflicts TCP. Namely, streams within a QUIC connection can be handled independently from one another, so that loss of packets of a stream does not have an impact on packets of other streams. Thirdly, TCP is not secure by default and hence needs TLS to run over it as an additional protocol. QUIC, instead, is an encrypted-by-default protocol that already includes TLS 1.3 handshake in its connection establishment process. Finally, QUIC provides a client-side connection migration mechanism which allows the connection to be kept after the client changes its IP address (e.g., due to a wireless handover). The proposed toolbox includes an extended version of the QUIC protocol to support the seamless migration of the service connection also when the server's IP address changes (e.g., which can occur upon a service migration) [53]. The extended protocol includes two strategies to support server-side connection migration in QUIC, which are called Explicit and Pool of Addresses. Both these strategies are based on the idea that server-side connection migration can be achieved by introducing minimal and non-breaking additions to both QUIC client and server, which are hence made aware of migration. The two strategies differ in the following respects. With the Explicit strategy, as the name says, QUIC server is explicitly informed at runtime of an imminent container migration and notifies QUIC client with the exact destination address (i.e., IP and/or port) right before migration starts. This explicit information speeds up connection migration, as the destination address is deterministically known. On the other hand, the Pool of Addresses strategy assumes that the container can be migrated within a predetermined set of server machines. During connection establishment, QUIC server notifies QUIC clients with the pool of possible destination addresses. With this strategy, QUIC does not deterministically know the next destination address, which should worsen overall performance. However, the Pool of Addresses strategy, unlike the Explicit one, allows to statically pre-define the pool of destination addresses, without the need for any interaction with the migration management at runtime. Both strategies have been integrated and tested in a real implementation of the protocol.

Finally, the toolbox can optionally be complemented by an edge computing-platform based on the ETSI Multi-access Edge Computing (MEC) standard. The latter standardizes an open

and multi-vendor edge-computing environment, to support the execution of services. An extension of the standard to support the migration of container-based MEC services is proposed in [54].

2.4.2 Nethuns: a library for efficient, socket-independent network I/O

N*ethuns* [55], a lightweight userspace library that offers a straightforward programming model for network I/O and can use several I/O accelerations frameworks, nicknamed engines, as a backend.
The general API defines the socket abstractions as points of access to a hardware interface queue, and ring abstractions, that are circular queues of packet descriptors. Each socket can have at most one ring per direction (tx or rx). Each ring descriptor contains packet metadata (e.g., time of arrival for incoming frames) and an in-use flag that is set/reset using atomic release/acquire memory operations as appropriate. With sockets open for rx, each call to the nethuns receive function returns the packet identifier of the next packet in the input stream. If the socket is non-blocking and no packet is available, an invalid identifier is returned, and the application is expected to try again later.
The function also returns a pointer to a packet header containing packet metadata and a pointer to a buffer with the full contents of the packet. The buffer must be later returned to the library, so that it can be reused to hold other incoming packets. Only one thread should call the receive function on each socket, but the received buffers can be released, in any order, by other threads. The memory necessary to hold the rings and the buffers for incoming/outgoing packets is managed by the library: users receive buffers from the library and return them when they are done with processing.
To transmit packets, the application may call the nethuns send function with a pointer to the packet and its size. The function only guarantees that the packet is queued up for transmission, but it does not necessarily wake up the underlying engine to actually send the packet. The application must call the nethuns flush function to actually pass the pending packets to the engine, which may then issue a batch transmission operation, if available in the hardware.
Applications built against the nethuns library must select an engine at compile time. This allows for compile time specialization of the most performance sensitive parts of the API, including the exact form of the nethuns rings and some access functions to packets and metadata that can be expanded inline. Porting the application to another system where a different engine is available requires only a recompilation. This design decision makes it hard to use two different engines at the same time in the same application, but it guarantees the best performance overall.
The effectiveness of the approach has been shown by porting Open vSwitch (OvS) to nethuns in a single man-day of work and running it with the AF_XDP and the netmap engines. The performance of the nethuns/AF_XDP version has been found to be comparable to the native AF_XDP implementation in OvS, while the nethuns/netmap version (obtained with a simple compilation switch) has been found to scale better. The results have been selected among the “Best papers of CCR” for the year 2022 and presented at SIGCOMM 2022. A nethuns “source” has also been added to WindFlow (see Section 2.2.6), to experiment with Stream Packet Processing, with good results.

2.4.3 CAPIO: cross-application programmable I/O

CAPIO (https://github.com/alpha-unito/capio) is a middleware file system in user space that boosts the performance of existing scientific workflows that communicate through files without modifying the original code. This tool allows users to coordinate the I/O and inject streaming capabilities into a workflow. CAPIO comprises two layers: the CAPIO runtime and

the CAPIO coordination language. The CAPIO runtime intercepts the I/O system call of the application. It optimizes the communication that otherwise would have leveraged the file system using the information expressed in the configuration file written with the coordination language. The CAPIO coordination language allows the user to write a configuration file enabling the CAPIO runtime to optimize the communication between applications transforming a batch execution into a streaming execution following the given streaming semantics and expressing in-situ and in-transit data transformations.

2.4.4 INSANE: A Uniform API for QoS-aware Host Networking as a Service

The Integrated aNd Selective Acceleration for the Network Edge (INSANE) middleware, is a general-purpose and lightweight userspace network stack that provides a uniform API to a wide range of network acceleration technologies, including XDP, DPDK, and RDMA, even available to the same supported application, with the goal of easing the development, deployment, and portability of latency-critical services in the cloud continuum. Following a microkernel-inspired architecture [56], INSANE consists of two main components:
A client library, exposing a uniform API with a minimal set of communication primitives, yet expressive enough to let developers define high-level and domain-specific abstractions on top of them. Through a set of Quality of Service (QoS) parameters, applications can define differentiated network requirements for their data flows, such as latency-sensitiveness, reliability, and resource consumption. Unlike in Demikernel [57], in our solution, flows with different requirements can be mapped to different technologies.
A runtime, working as a network stack as a service for applications and offering common services for high-performance networking, including memory management for zero-copy transfers, efficient packet processing, and different packet scheduling strategies. A plugin-based architecture allows the specialization of such abstractions for each integrated network acceleration technology. In particular, the high-level QoS requirements specified by applications are used to dynamically map the flows to the most appropriate acceleration technology that is dynamically available at the deployment site.

2.5 Hardware layer

2.5.1 MLIR: multi-level intermediate representations for heterogeneous computing platform

The multi-level intermediate language (MLIR) [58] is an extension of the LLVM toolchain recently introduced to make compiler-level code optimizations more flexible. 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 during program translation.
Adopting MLIR enables to build reusable and extensible compiler infrastructures for specific application domains. Overall, this approach can address software fragmentation by connecting multiple software tools together, significantly reducing the cost of building domain-specific compilers.
In this project, a standard MLIR-based workflow will be extended to introduce new dialects for domain-specific HPC applications (e.g., sparse linear algebra [59], training and deployment of deep neural networks [60]) with the aim to target heterogeneous architectures such as the RISC-V accelerators proposed by Flagship 2. The project activities will deliver a prototype consisting of an MLIR-based tool supporting a RISC-V core with instruction set architecture (ISA) extensions designed to improve the performance of the HPC workloads mentioned above. The proposed tool will perform a progressive lowering of the

high-level abstractions down to the target architecture, with the aim of exploiting the ISA extensions without modifying the original code.

3 State of the Art of the tools for the Hybrid Cloud-HPC architecture

While the previous Section introduced the HPC tools developed by each partner, this Section provides an overview of the advancements made by each tool, highlighting their unique features and contributions to the field of high-performance computing, positioning each tool with respect to its current state of the art. This will pave the way for the GAP analysis presented in the next Section.

3.1 Application layer

3.1.1 BDMaaS+

The management of complex IT services installed on federated and hybrid cloud environments is important to distribute the workload of such services on the underlying IT architecture. Hybrid Cloud-HPC refers to computing scenarios with a mix of public and private heterogeneous computing resources usually deployed in distant locations. The Hybrid Cloud-HPC can bring many benefits to service providers that want to optimize their IT architecture, by moving some components to different Cloud data centers to gain access to more powerful resources or to reduce the operational costs for service provisioning. However, the high sophistication of modern IT services and the complexity of Hybrid Cloud- HPC environments make it extremely difficult, at the business level, to evaluate the impact of changes to an IT service architecture before deploying them. The performance assessment of possible alternative deployments calls for service management tools that provide what-if scenario analysis functions capable of exploring multiple IT service configurations, evaluating their performance through a comprehensive business-level behavioral analysis, with the purpose of identifying the most convenient one.

3.1.2 WorldDynamics

In the early 1970s, Jay Forrester, a notable and pioneer researcher in several fields, designed the first system dynamics model intended to analyze the complex interactions between high-level subsystems of the world, to understand the evolution of a major variable in each subsystem (in particular, the population, the capital investment, the natural resources, the fraction of capital devoted to agriculture, and the pollution level variables). His model, called World2, was published in the 1971 book "World Dynamics" [61] and subsequently evolved into the World3 model, described in the well-known 1972 book "The Limits to Growth" by Meadows et al. [10]. Since then, several similar models were developed, such as the quite popular DICE model from the Nobel Prize laureate William Nordhaus, and categorized as Integrated Assessment Models (IAMs), which can be roughly divided into two general classes: policy optimization and policy evaluation models. DICE is a policy optimization model in which, intuitively, a set of parameters (that is, a policy) which maximizes a specific objective variable (or function) are chosen. On the other hand, the WorldX models (which recently evolved in the Earth4All model [62]) can be considered as policy evaluation models, in which a policy is decided (that is, a set of parameter values) and the behaviour of some major variables (or functions) is analyzed. Benefiting from Julia's ecosystem for scientific computing and referring to this classification, *WorldDynamics.jl* [9] (which has already been applied to the reproduction of several historical models) will be mostly developed to support the construction of policy evaluation IAMs.

3.1.3 Real Time Simulator for Digital Twin and Hardware-In-Loop in the Electrical Power Networks Scenario

The normal and correct operation of the power network is achievable and can be maintained only with proper monitoring infrastructure. The kind of infrastructure is related to: (i) the economic availability of the system operator (SO); (ii) the portion of the considered network; (iii) the quantities to be measured and the desired target accuracy. Target uncertainty must be fixed during the design and development stage of a DMS thus allowing to define the minimum requirements that the monitoring devices must have. This is a critical choice because the final cost of the single measurement unit will significantly differ (even several orders of magnitude) depending on it. In recent years, the above studies have been supported by new tools for network monitoring: real-time simulators (RTS). Such tools allow SOs and researchers to simulate and run portions or complete networks with the aim of better understanding them and predicting their behavior. To work properly and to avoid incoherent and wrong results, RTSs need models for the assets and elements of the grid, which consider nominal and off-nominal operations, and accurate as well as realistic generation and load profiles. RTS systems enable different types of simulations, and the most promising techniques are hardware-in-the-loop (HIL) and the digital twin (DT). The former technique consists of exploiting the analog/digital input/outputs of the RTS to include real devices inside the simulation. This way, the information from the physical world can be included in the virtual environment. As for the DT, instead, it enhances the previous concept, upgrading it to the level at which the information obtained in the virtual world is used to improve the physical one, and vice versa. This means that there is a real-time interaction and mutual information exchange between the virtual and physical worlds. It is of paramount importance noting that RTS systems must also be characterized and reliable enough according to the target uncertainty. On the contrary, the risk is to include in the measurement system or in the DT environment an element in which characteristics are far worse than those of the other devices.

3.1.4 Interactive Computing Service

The common approach to access HPC resources is via the job batch scheduler, even if different ones can be put in place to allow more user-friendly and interactive actions. The focus is on the deployment of a framework, named “InterActive Computing” (IAC), which is accessible via a web browser interface, and allows interactive sessions on an HPC cluster. It is based on the “Gaia” [63] platform developed by “E4 Computer Engineering” company, and it is aimed to provide near-instantaneous access to HPC computational resources. The implementation is realized as an ad hoc software stack relying on Jupyterlab [64] as the main component. Additional custom features with respect to the standard Jupyterlab implementation are put in place in order to improve the workflow management approach for a more user-friendly experience. The IAC framework represents a very flexible solution for a wide range of scientific areas which demands a responsive resource allocation, a non-static workflow and/or an on-the-fly data visualization tool. Even if the IAC approach is natively suitable for Artificial Intelligence, machine learning and deep learning applications, it can be extended to a broad spectrum of HPC fields such as Data Analysis, Quantum Computing emulations, and many more, since several tools suitable for such fields are already distributed in a form that can be easily integrated within such IAC framework. The platform can also act as development infrastructure for some of the use cases proposed and developed in the context of the National Center for HPC, Big Data and Quantum Computing, resulting in a clear opportunity to have a multidisciplinary cooperation.

3.1.5 Jupyter Workflow

Scientific workflows and HPC communities are converging on the same objectives to provide effective workflow management in a combined HPC and distributed environment [65]. Jupyter is an ideal tool for both communities. Given their widespread diffusion, Jupyter Notebooks [66] have already been investigated to bridge the gap between non-IT practitioners and HPC infrastructures to make interactive workflows mainstream in HPC centers. As an example, the National Energy Research Scientific Computing Center (NERSC) adopts Jupyter as an interface to the CORI supercomputer [67], and the PANGEO platform component for HPC is based on the integration between Jupyter and Dask [68]. The reason for their success is their capability to support step-by-step execution and interactive tuning of software pipelines, boosting the productivity of scientists. A second reason is their portability, which is a prerequisite for reproducibility. However, the lack of support for complex workflows and the challenging integration with hybrid Cloud-HPC architectures undoubtedly hampered their adoption in production workloads.

3.1.6 StreamFlow

When considering data-intensive scientific workflows, all data management aspects become crucial. For example, in situ data processing strategies can prevent all the overheads related to data transfers and even optimize disk I/O when either in-memory processing or burst buffers are available. Conversely, distributed executions become mandatory when dealing with federated data access or strict privacy policies. Therefore, a modern Workflow Management System (WMS) should be capable of orchestrating hybrid workflows, i.e., coordinating tasks running on different execution environments [69]. Many grid-native WMSs (e.g., Askalon [70], Pegasus [71], Taverna [72], Triana [73], and Galaxy [74]) support distributed workflows. However, each WMS adopts its technological stack for the communication layer, often relying on low-level external libraries that must be installed and pre-configured on each node involved in the workflow execution (e.g., GAP [75], GLARE [76], HTCondor [77]). A modern WMS should also automatically orchestrate the execution environment's life cycle, providing support for complex container-based deployments to guarantee portability and reproducibility and potentially scaling the available resources according to the workflow needs. Along this line, WMSs and orchestration technologies can benefit from each other. For example, several workflow frameworks have been built natively on top of Kubernetes (e.g., Pachyderm [78], Argo (https://argoproj.github.io/)), and the learning cloud vendors are currently focusing on offering hybrid solutions to combine multi-cloud and on-premises infrastructures (e.g., GoogleCloudComposer (https://cloud.google.com/composer/), based on Apache Airflow (https://airflow.apache.org/)). Despite offering great flexibility in interacting with Kubernetes resources, these products are tightly coupled with such technology and do not allow task offloading on different environments, such as HPC sites.

3.1.7 Parallel Multi-density Clustering

In the field of urban data analysis, the detection of city (or urban) hotspots is becoming a more and more popular task. Given the availability of geo-referenced data, urban hotspots can be considered as dense regions in spatial data, serving as a valuable organization technique for framing detailed knowledge of a metropolitan area. They provide high-level summaries for spatial datasets, which are valuable knowledge to support planner, scientist, and policymaker's decisions. Among several spatial analysis approaches, classic density-based clustering algorithms have been shown to be very suitable to detect urban hotspots in a city [79]. In particular, urban hotspots are detected in the form of density-based regions, i.e., areas in which urban events (i.e., pollution peaks, viral infections, traffic spikes, crimes) occur with a higher density than in the remainder of the dataset. Such algorithms also satisfy

several properties that are usually required for spatial clustering algorithms. However, due to the adoption of global parameters, they fail to identify multi-density hotspots (i.e., characterized by varied densities), unless the clusters (or hotspots) are clearly separated by sparse regions. This is a crucial issue when analyzing urban data, because the density of population, traffic, or events in cities can vary widely from one area to another area. In particular, metropolitan cities are extremely dissimilar urban regions in terms of density. For such a reason, multi-density clustering approaches show higher effectiveness to discover city hotspots. Moreover, the growing volumes of data collected in urban environments require high-performance computing solutions, to guarantee efficient, scalable and elastic task executions.

3.1.8 aMLLibrary

Machine Learning (ML) has been widely applied to predict the performance of several kinds of Information and Communications Technology (ICT) systems. A first example is video streaming network platforms, which attempt to infer the actual quality of service starting from measurements of some Quality of Delivery (QoD) metrics [80]. Other domains in which ML models are commonly leveraged for performance prediction include cloud systems, Artificial Intelligence (AI) models, communicating networks, and Functions as a Service (FaaS) systems. For instance, [81] examines the performance of several ML models in carrying out predictions of execution times of Apache Spark jobs with different types of workloads. Their results outperform models used by Spark creators. [82] proposes a ML-based prediction platform for Spark SQL queries and ML applications, which exploits features related to each stage of the Spark application, as well as previous knowledge of the application profile. [83] employs several ML models alongside anomaly detection to properly configure a cloud-based Internet of Things (IoT) device manager while respecting Quality of Service (QoS) constraints. [84] explores performance prediction of training times of GPU-deployed neural networks starting from software-hardware specifications, by using ML techniques and feature selection methods. Similarly, [85] compares some popular ML techniques applied to a workload prediction analysis on HTTP servers, showing that these techniques all achieve good predicting capabilities. The underlying algorithm exploits the predicting capabilities of ML models by integrating them with Bayesian Optimization (BO) methods. The Schedulix framework [86] uses linear regression to estimate execution latencies of serverless applications in a public cloud FaaS setting. Finally, [87] proposes the PrePass-Flow technique for the context of hybrid Software-Defined Networking (DSN) architectures, where failure of legacy network nodes is communicated with a delay. In particular, their framework uses ML models such as logistic regression and Support Vector Machine (SVM) to predict such failures before their occurrence.

3.2 Static optimization and transformation layer

3.2.1 Sieve, Process, and Forward (SPF)

Service and resource management across the Compute Continuum is a compelling task that requires novel methodologies and tools capable of orchestrating the processing of the deluge of data generated by IoT sensors by exploring adaptive and lossy methodologies. In this regard, information-centric concepts can provide substantial advantages in defining service composition and orchestration. Specifically, the definition of information-centric and composable services, which could be dynamically migrated, e.g., from fog to the cloud in case more processing power is required, could well suit the task. Composable services allow the definition of fine-grained service components that can be arranged together based on the

kind of information they consume and produce. This is different from solutions such as Web Services Description Language (WSDL) and Business Process Execution Language (BPEL) which instead consider the service API and location.

3.2.2 ParSoDA: Parallel Library for Big Data Analysis

Big data analysis aims at extracting useful knowledge from very large amounts of data gathered from different, and often heterogeneous, sources. In most cases, the amount of data to be analyzed is so big that high-performance computers, such as many and multi-core systems, Clouds, and multi-clusters, paired with parallel and distributed algorithms, are used by data analysts to reduce response time to a reasonable value. Several developers and researchers are working on the design and implementation of tools and algorithms for extracting useful information from big data. In such cases, the use of parallel and distributed data analysis techniques, frameworks and programming models (e.g., MapReduce) is essential to cope with the size and complexity of data to analyze. However, it is hard for many users to use such solutions for tackling big data issues, mainly due to the programming skills needed for implementing the appropriate data analysis methods on top of complex distributed systems and runtimes. Several research projects consider not only the data analysis task, but also procedures including other data processing tasks needed for building data analysis applications. In particular, these projects aim at helping scientists to implement all the steps that compose data mining applications without the need to implement common operations from scratch. Some existing work focused on designing and optimizing data preparation [88], data extraction and analysis [89] [90], data analytics [91], but most of them do not support Clouds and/or multiple distributed runtimes at the same time.
Differently from such systems, ParSoDA [24] was specifically designed to implement parallel scalable data analysis applications, mainly focusing on data gathered from social media. To this end, it provides scalability mechanisms based on two of the most popular parallel processing frameworks (Hadoop and Spark), which are fundamental to provide efficient and scalable services as the amount of data to be managed grows.

3.2.3 BLEST-ML (BLocksize ESTimation via Machine Learning)

Data partitioning refers to splitting a dataset into small and fixed-size units, called blocks or chunks, to enable efficient data-parallel processing and storing in distributed-memory-based systems. Several issues related to data partitioning must be addressed to reduce execution times and ensure the good scalability of applications. For example, when a dataset is mapped on a set of nodes of a parallel/distributed computing system, two very critical problems are highlighted: (i) the choice of the destination node for a given block (i.e., the node where that block will be stored); and (ii) the selection of an appropriate block size. The first problem has been widely addressed through the proposal of several scheduling algorithms aimed at minimizing the movement of data at run-time. The second problem instead, less studied in the literature, requires taking a decision before the application is running as it strongly depends on the features of the input dataset, the algorithm, and the execution environment.
The block size can heavily affect the trade-off between single-node efficiency and parallelism in data-intensive applications. Specifically, a larger size reduces parallelism (fewer blocks) but makes tasks larger. Although this can lead to an overhead reduction, it must be ensured that the block size does not exceed the memory available on the individual nodes, to avoid memory saturation. On the other hand, a smaller size leads to finer exploitation of parallelism, while introducing a larger overhead due to communication, synchronization, and task management, which can negatively impact performance. Typically, block size estimation is not an easy task for programmers. In fact, they usually proceed by following a trial-and-error approach, only supported by simple heuristics and domain knowledge. As a result, this

tuning process is often time-consuming and resource-intensive, especially when large datasets and complex hardware infrastructures are used.
To address this issue, BLEST-ML (BLocksize ESTimation via Machine Learning) [27] is proposed. BLEST-ML is a novel methodology that leverages a machine learning-based approach to determine a suitable estimate of data block size for hybrid partitioning, thus optimizing the execution of data-parallel applications on large scale high-performance infrastructures, by requiring minimal resources and domain knowledge.

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

Compressing large collections of files is a very well-known problem that 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 Manber [92] and Heintze [93] 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 et alii [94] and Charikar [95].
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 adding the projections of all its tokens. Comparing these two fingerprinting approaches it can be noticed that SimHash has been designed to take into account only the frequency of tokens; conversely, MinHash has been designed to consider the tokens order (via shingling) but not their frequency. Nevertheless, both of them can be adapted to offer all these features. 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). A number of papers (see e.g., [96], [97], [98]) have compared these two fingerprinting methods 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 [99], which were proved to achieve better mathematical guarantees than SimHash.
Coming back to the problem of compressing large collections of HTML-files, 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 [28] they addressed the problem at hand via the so-called PPC paradigm: Permuting + Partition + Compress, whose main algorithmic idea is to first permute the files in order 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).

3.2.5 MALAGA, MultidimensionAL Big DAta Analytics over Massive Graph DAta

Massive graph data arise in a plethora of big data application settings, ranging from bioinformatics to smart cities, from social network analysis to intelligent transportation

systems, and so forth. In this so-delineated scenario, supporting big data analytics over massive graph data is an emerging research challenge in the big data research context, with also surprising presence of inspiring platforms and technologies coming from the industry. Looking at active literature, there exist several research proposals that focus the attention on the issue of supporting big data analytics over massive graph data (e.g., [100], [101]). For instance, [102] proposes *GraphH*, a system to support high-performance big graph analytics in small clusters. [103] studies the problem of capturing and querying provenance from high-performance graph data processing systems and proposes a batch-processing tool called *Ariadne*. [104] introduces *I-HASTREAM*, a density-based hierarchical clustering algorithm over time-variant big data streams for big graph analytics purposes.
Multidimensional big data analytics ([105], [106], [107]) positions itself as a state-of-the-art big data analytics paradigm where the main innovation consists in embedding fortunate multidimensional metaphors [108] into the target big data analytics process. Basically, this paradigm pursues the idea of modelling both the target big dataset and the big data process itself in terms of *dimensions* (of analysis) and *measures* (of interest), according to the underlying big data analytics goals. This contributes to magnify the expressive power and to achieve a relevant gain into the discovery of actionable-knowledge insights.

3.2.6 FastFlow/WindFlow: high-level and efficient streaming

Parallel programming in the whole HPC continuum scenario still relies on low-level programming frameworks and a massive number of machine-dependent optimizations. In most scientific workflows running on supercomputers, the de facto standard parallel programming model is MPI used with OpenMP or CUDA to accelerate local-node kernels. Such a mix of low-level and quite different programming models still makes parallel programming a niche for expert programmers.
In the field of Big Data Analytics, Data Stream Processing has played an essential role as an enabling computing paradigm to process unbounded data streams. The first systems of this kind were Data Stream Management Systems (DSMSs) [109] proposed as extensions of traditional DBMSs, where streams were considered as special classes of relations and applications were developed as queries expressed through SQL-like formalisms. More recently, DSMSs have evolved into scale-out general Stream Processing Engines (e.g., Apache Storm, Flink, Spark Streaming, Samza, S4), which provide higher-level programming interfaces and fault-tolerant runtime systems to distribute streaming applications on Cloud infrastructures.
As the boundaries between HPC and Big Data Analytics continue to blur, data streaming capabilities will represent the key features of any parallel programming model.

3.2.7 Clustering Algorithm

Clustering algorithms are efficient tools for discovering correlations or affinities within large datasets and are the basis of several Artificial Intelligence processes based on data generated by sensor networks. Recently, such algorithms have found an active application area closely correlated to the Edge Computing paradigm. The final aim is to transfer intelligence and decision-making ability near the edge of the sensor networks, thus avoiding the stringent requests for low-latency and large-bandwidth networks typical of the Cloud Computing model. The present tool is based on a new hybrid version of a clustering algorithm for the NVIDIA Jetson Nano board by integrating two different parallel strategies. The algorithm is aimed to improve the relationship between performance and energy consumption. First results confirm the possibility of creating intelligent sensor networks where decisions are taken at the data collection points [33].

3.2.8 High performance and Low Power Hyperspectral Image Analysis

Hyperspectral Imaging is a powerful technique allowing the acquisition of detailed spectral information of a scene. It collects data across a wide range of wavelengths in the electromagnetic spectrum. This data creates a "spectral signature" for each pixel in an image, used to identify and classify different materials and substances within the scene. One of the significant advantages of Hyperspectral Imaging is its ability to detect and identify invisible materials to the human eye or traditional imaging systems. This characteristic makes it an ideal tool for various applications, including mineral and oil exploration, environmental monitoring, and military surveillance. However, Hyperspectral Imaging can be computationally costly, as it involves collecting and processing large amounts of data across a wide range of wavelengths. This operation often requires high-performance computing resources, such as large amounts of memory, high-speed storage, and parallel processing capabilities. Furthermore, it should be considered that the issue of energy consumption in computing infrastructures has become increasingly stringent. This is true both for large data centers as well as for low-power devices characteristic of the edge computing model. The achievement of a compromise between high performance and low power consumption is therefore of fundamental importance for the entire HPC community [110].

3.2.9 DivExplorer: Analyzing Machine Learning Model Behavior via Pattern Divergence

Machine learning models and automated decision-making procedures are becoming more and more pervasive. The evaluation of their 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. Typically, domain expert help is required to identify relevant (or sensitive) subgroups.
Several existing approaches that explore differences in subgroup performance [111], [112] require users to specify the attributes or attribute values of interest. This 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 [113] adopt clustering techniques. However, the identified clusters are not directly interpretable, limiting the actionable understanding. Approaches as [114], [115] 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 [114] or are optimized only to derive subgroups with lower performance than the average, not allowing for a complete understanding of the model behavior.

3.3 Orchestration layer

3.3.1 BookedSlurm

Accessing HPC resources in a shared environment is usually done through a job scheduler. Jobs are submitted to a queue and scheduled when the requested resources are available, considering a more comprehensive range of parameters such as job priority, age and size, and the possibility of backfilling without impacting the general FIFO queue. The Simple Linux Utility for Resource Management (Slurm) [116] is a job scheduler which allows users to allocate resources for a defined time, providing a framework to start and monitor jobs and maintaining a queue of pending jobs to be scheduled. Slurm can also collect accounting information for the scheduled jobs, keeping a history of used resources and execution time. Every trackable resource can be suitably weighted for custom needs. After each job execution, a related "billing" value is calculated by adding up the weighted uses previously defined, giving an estimated total cost of the job, which has no direct use or implications apart from accounting and statistics.

3.3.2 Orchestration of composite containerized applications in the Cloud continuum

Cloud resource orchestration denotes various processes and services to select, describe, configure, deploy, monitor and control cloud resources across different cloud solutions in an automated way. The overall goal of cloud orchestration is to guarantee successful hosting and seamless delivery of applications by meeting the Quality of Service (QoS) goals of both cloud application owners and cloud resource providers. Many cloud industry players have developed cloud management platforms to automate the provisioning of cloud services (e.g. Amazon CloudFormation (https://aws.amazon.com/cloudformation/), Flexera Cloud Management Platform (https://www.flexera.com/products/agility/cloud-management-platform.html), RedHat CloudForms (https://access.redhat.com/products/red-hat-cloudforms), IBM Cloud Orchestrator (https://www.ibm.com/us-en/marketplace/deployment-automation)). The most advanced platforms also offer life-cycle management of cloud applications. These commercial products are neither open to the community nor portable across third-party providers. Cloud applications natively designed and built to run on a Cloud platform are basically locked in that platform, i.e., the effort to port the application to a different cloud platform is often not worthwhile. Simply put, by cloud application portability it is usually meant the capability of a cloud application to be easily and seamlessly ported across different and heterogeneous cloud platforms. In order for a cloud application to be classified as portable, the operations related to the management of its life cycle (which range from the very first deployment to the final disposal) should not depend on the specific cloud platform(s) that may be supporting it. TOSCA [7] is a standard designed by OASIS to enable the portability of cloud applications and the related IT services in a way that is vendor-agnostic. In the TOSCA specification, the structure of a cloud application is described as a service template, which is composed of a topology template and the types needed to build such a template. TOSCA defines a few normative workflows (deploy, undeploy, configure, start, scaling, and auto-healing) to operate a topology and specifies how they are declaratively generated. In the literature, as well as in the commercial setting, quite a number of proposals have appeared that have embraced the TOSCA specification to offer platforms supporting the design, deployment and management of portable cloud applications.
Cloudify [117] is a TOSCA-compliant open-source orchestration framework that provides services to model applications and automate their entire life-cycle through a set of built-in workflows. In Cloudify, cloud application templates are referred to as blueprints, which are YAML documents written in Cloudify's Domain Specific Language (DSL). Despite being aligned

with the modeling standard, Cloudify's DSL does not directly reference the TOSCA standard types. MiCADO (Microservices-based Cloud Application-level Dynamic Orchestrator) [118] is an open-source multi-cloud orchestration and auto-scaling framework for Docker containers, orchestrated by Kubernetes (or alternatively by Docker Swarm). MiCADO supports multi-cloud and cross-cloud deployments on various public and private cloud infrastructures. It provides interoperability and portability by means of a TOSCA-based Application Description Template (ADT). INDIGO-DataCloud (INtegrating Distributed data Infrastructures for Global ExplOitation) [119] is an open-source data and computing platform targeted at scientific communities, and provisioned over Cloud and Grid-based infrastructures as well as over HTC and HPC clusters. INDIGO provides automatic distribution of applications and/or services over a hybrid and heterogeneous set of IaaS infrastructure. It supports multi-cloud and cross-cloud deployments, as well as interoperability by leveraging open standards (OCCI, CDMI). It also promotes portability by adopting an extension of TOSCA for describing applications and services.

3.3.3 Liqo

Despite the emergence of common interfaces for applications orchestration being key towards a real edge to cloud continuum [120], [121], industry-standard approaches 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. Indeed, edge data centers cannot depend on a single centralized control plane, for resiliency (i.e., preventing failure propagation in case of network partitioning) and performance reasons, as orchestration platforms typically suffer if nodes are geographically spread over high-latency WANs [122], [123], [124]. Besides the edge landscape, resource fragmentation also affects larger data centers, with many companies increasingly witnessing the cluster sprawl phenomenon [125], [126]. This trend finds its roots in scalability concerns, in the hybrid-cloud (i.e., the combination of on premise and public cloud) and multi-cloud approaches [127], which aim for high availability, geographical distribution and cost-effectiveness, while granting access to the breadth of capabilities offered by competing cloud providers. Additionally, non-technical requirements such as law regulations, mergers and acquisitions, physical isolation policies and separation of concerns contribute to the proliferation of clusters. Fragmentation also hinders the potential dynamism in the workload placement [128], [129], [130], forcing each application to be assigned upfront to a specific infrastructure. No resource compensation is ever possible, hence preventing jobs from transparently moving from an overloaded cluster, e.g., due to unexpected spikes of requests, to another one, underused and potentially offering better performance. At the same time, the deployment of complex applications composed of multiple microservices, each one with specific requirements (e.g., low latency, high computational power, access to specialized hardware), as well as the enforcement of proper geographical distribution and high-availability policies, requires the interaction with different infrastructures. However, this prevents relying on the single point of control abstraction, which would allow the deployment of arbitrarily complex applications across the entire resource continuum, no matter how many nodes and clusters it is composed of.

3.3.4 Energy efficient orchestration and resource management in the cloud continuum

Virtual machine (VM) orchestration and platform resource management in cloud systems is a well-investigated subject, as it is a crucial task to ensure the quality of service (QoS) required by the services and applications running on the VMs on cloud platforms.

The placement of VMs can be compared to a multi-dimensional variable-sized bin-packing problem, in which the CPU, memory, disk, and network bandwidth of the host are part of the constraints including the requirements of the VMs. As demonstrated in the literature, the problem is NP-hard, consequently, a wide range of near-optimal solutions have been proposed.
Among the objectives of the proposed solutions, recent energy-efficient approaches have been introduced to reduce the energy footprint of cloud infrastructures, still orchestrating VMs and allocating resources to enforce application QoS requirements.

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

Existing open-source FaaS frameworks (e.g., OpenFaaS and OpenWhisk) are not well suited for Edge environments for a number of reasons: they use centralized schedulers or gateway components, which introduce latency in geo-distributed settings; they rely on memory-demanding function sandboxes, usually based on software containers; and they support only 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 [50] and Sledge [51]). However, these solutions either work within single Edge nodes or scale over multiple nodes without considering geographical distribution.
Serverledge is a decentralized FaaS platform designed from scratch for Edge-Cloud computing environments. Its architecture consists of one or more nodes that can be deployed in Cloud data centers and/or at the edge of the network and spread across multiple zones, and a global registry. 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 Serverledge node, thus not requiring reaching a centralized gateway, possibly distant, for scheduling.
Because of the limited resource capacity of Edge nodes, it is likely that a single node or multiple nodes located in the same zone cannot sustain the incoming load. Therefore, Serverledge supports vertical (i.e., from Edge to Cloud) and horizontal (i.e., among Edge nodes) computation offloading.
Serverledge also supports the definition of QoS-aware scheduling and offloading policies that take into account user-specified QoS requirements (e.g., maximum response time) for the function invocation request.

3.4 Runtime management layer

3.4.1 MoveQUIC: a QUIC-based toolbox for the live migration of microservices at the network edge

Edge computing is strongly emerging as an extension of cloud computing towards the network edge. With edge computing, microservices can run in a pervasive infrastructure of geo-distributed micro data centers that are located close to (or co-located with) access networks [131]. Providing edge-hosted microservices has a great potential but, at the same time, reveals some issues. First, mobile end users at the edge may roam across different network access points. When this occurs, microservices accessed by the user may have to migrate among edge servers in different micro data centers to maintain proximity to the client. Second, to balance the load during high-load periods or to reduce the number of active

hosts during low-load periods, microservices could be moved across edge data centers, thus enabling more dynamic power-saving mechanisms [132]. In scenarios such as those presented above, several actions may need to be taken to guarantee service continuity. On the one hand, the state of the microservice should be maintained across edge servers. On the other hand, service communication should be maintained as much as possible transparently to the end-user and to the application [133].

3.4.2 Nethuns: a library for efficient, socket-independent network I/O

Programmability plays a central role in the datacenter network path. Hardware solutions based on P4 [134] are popular, but traditional commodity servers are still effective, thanks to their flexibility [135], [136], [137], [138] [139]. However, the standard networking stack shipped with popular Operating Systems such as Linux is unable to attain top class performance, especially when high packet-rates are needed. The software approach, therefore, requires the adoption of accelerated network I/O frameworks such as DPDK, netmap and AF_XDP. Each one of these frameworks is characterized by its own programming model and incompatible API. Programmers must typically target only one of these frameworks, hindering portability.

3.4.3 CAPIO: cross-application programmable I/O

In the exascale era, the gap between the computation speed and the speed of the I/O operations is increasing. Much has been done to try to reduce this gap, from building APIs that provide collective I/O (MPI I/O) operations to new ad-hoc file systems [140]. Much has also been done to reduce the workload of data format expression (e.g., HDF5). Previous studies show that these works are not used extensively by the community. Potential reasons may be that these tools are not easy to master and that many scientific workflows are composed of legacy code that nobody wants to touch. As shown in [141], [142], [143], most applications use the POSIX file API to communicate with each other.
CAPIO aims to reduce the I/O time and reduce the burden on the programmer by providing a file system in user space that can boost scientific workflow composed by the application that communicates through files without modifying the original code. This file system in user space is also programmable, which means that the user can use this tool also to create a new scientific workflow using files for communications (UNIX philosophy) and, in a second moment, provide the semantics that expresses the type of communication.

3.4.4 INSANE: A Uniform API for QoS-aware Host Networking as a Service

The ability to process and analyze data under stringent time constraints is quickly becoming a key requirement of modern data-driven applications, leading to a growing demand for systems that can achieve ms-scale latencies while maintaining high levels of reliability, scalability, and efficiency [57], [144]. To avoid becoming a bottleneck, widely popular systems, ranging from key-value stores to state-machine replication engines, have been carefully re-designed to leverage kernel-bypassing I/O techniques and modern hardware that implements common operating system services, such as host networking, in more efficient way. An even more radical approach to reduce service latencies is emerging under the paradigm of edge cloud computing, which combines such acceleration techniques with the idea of moving components on small-scale datacenter-like environments physically close to datasources, in support of latency-critical services (e.g., Multi-access Edge Computing platforms, or MEC, for 5G operators).
Although kernel-bypassing techniques coupled with modern hardware have proved effective to achieve time-sensitive data processing, two critical concerns still prevent their wide-scale adoption. On the one hand, existing techniques for datapath acceleration require the use of custom and usually low-level interfaces that make application development difficult and

time-consuming, requiring advanced system expertise: not only applications must be re-architected and carefully optimized to fully leverage the performance benefits of acceleration technologies [145], but developers must also deal with the continuous release of updated device features and the concurrent deployment of different generations of hardware [56]. On the other hand, there are several possible techniques for host networking accelerations, which offload typical kernel tasks to user space or even hardware: the Linux eXpress Data Path (XDP), the Data Plane Development Kit (DPDK), or Remote Direct Memory Access (RDMA). Each of these options defines its own programming abstractions, network access interface, and memory management: hence, application portability is very hard to provide, and accelerated services can be deployed only onto a single and highly homogeneous environment, whereas real-world platforms are usually quite the opposite, in particular in the so-called cloud continuum [144]. For instance, RDMA is only intermittently supported by major public clouds. Even more importantly, the increasingly popular cloud edge computing platforms are highly heterogeneous in terms of software and hardware resource availability, making such lack of portability even more troubling right where latency-aware applications are most needed.

3.5 Hardware layer

3.5.1 MLIR: multi-level intermediate representations for heterogeneous computing platform

Modern HPC systems include a wide range of hardware architectures, such as CPUs, GPUs, FPGAs, and application-specific accelerators. Nowadays, programming 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 optimize code and achieve optimal workload mapping keeping the level of abstraction as high as possible [146].
In the last thirty years, three leading players have dominated the compiler market for HPC systems: GNU Project, Intel Corporation, and Portland Group (later acquired by NVdia). The compilers provided by Intel (e.g., icc and ifort) are optimized to exploit the features of x86 microprocessors to their fullest extent, resulting in higher performance compared to other alternatives for this target but inappropriate for heterogeneous platforms. In addition, all the compilers used in HPC systems provide support for the OpenMP and MPI programming models, typically coupled to manage a shared-memory multi-core computing environment within each node (OpenMP) and distributed message-based computation (MPI) across nodes.
Most recently, the LLVM compiler [147] infrastructure has been increasingly adopted in the HPC domain. LLVM has become an integral part of the software-development ecosystem for optimizing compilers, dynamic-language execution engines, source-code analysis and transformation tools, debuggers, and linkers. NVIDIA has adopted LLVM for developing its CUDA framework; following this trend, many other companies have contributed to LLVM development, pushing their work into its open-source codebase. The spread of LLVM has fostered the emergence of programming paradigms that enable practices and optimizations that are application-domain specific.
Designed as a state-of-the-art compiler toolchain, LLVM is a modular framework including three macro blocks: a front-end, a middle-end, and a back-end. The LLVM front-end (called clang) includes a set of software tools that recognize legal programs written in high-level programming languages (e.g., C, C++, and Fortran) and produce an intermediate representation (IR) for the following stages. IRs are fundamental in modem compilers since they convey information through the compilation process, enabling a wide spectrum of optimizations. LLVM IR is the intermediate representation adopted in the LLVM middle-end to facilitate architecture-agnostic optimization passes. Each middle-end pass transforms a program representation into an equivalent one optimized for a target metric (e.g., speed, size, or safety), and the design of LLVM IR simplifies this specific goal. Finally, the LLVM back-end includes a set of architecture-specific optimizations and produces binary code for the target machine. Nowadays, LLVM is the reference toolchain for compiler construction in both academia and industry.

4 GAP analysis of the proposed tools compared to HPC

In this Section, a basic GAP analysis for each tool is presented, with the aim to identify in which directions they should be improved, in order to foster the adoption of such tools for HPC applications. Overall, the goal of this Section is to identify areas of improvement for HPC tools through a GAP analysis. The aim of understanding the gaps in the current tools, is at focusing the future activities of the Flagship partners towards improving them and ultimately increasing the integration of different HPC tools and their adoption for HPC applications.

4.1 Application layer

4.1.1 BDMaaS+

Compared to the related efforts that mainly focus on single aspects of IT service optimization such as load-balancing-related objectives, Service Level Agreement (SLA) compliance, and cost minimization, BDMaaS+ adopts a comprehensive approach that takes into account all these different aspects from a business-driven perspective. Specifically, BDMaaS+ leverages a comprehensive service cost model that, beyond virtual resource acquisition, also considers Service Level Objective (SLO) violations and risk-related aspects. Differently from existing efforts that address the IT service problems by proposing mathematical models which require huge modeling simplification. BDMaaS+ adopts a simulative approach to reenact IT services under different configurations to accurately capture the peculiar behavior of real-life IT services, and it adopts an innovative optimization solution based on a memetic algorithm for enabling robust and resilient exploration of large and complex search space, thus realizing an effective what-if scenario analysis tool. However, BDMaaS+ adopts a pure Infrastructure as a Service (IaaS) perspective that mainly focuses on VMs. To take advantage of modern orchestration solutions such as Kubernetes, it would be interesting to extend the BDMaaS+ model to support state-of-the-art containerized applications. Therefore, BDMaaS+ could be used to analyze the impact of container-based HPC applications and to find the best placement of computing-intensive workflows in a federation of Kubernetes clusters.

4.1.2 WorldDynamics

WorldDynamics.jl will allow a designer to focus on a specific subsystem without necessarily knowing how another person is developing a different subsystem (interestingly, this seems to be how the well-known World3 model was described and, most likely, developed). Moreover, this approach will easily allow the substitution of one system of equations (representing one subsystem) with another system of equations (still representing the same subsystem), if it correctly interacts with the other subsystems (intuitively, respecting the required input/output variable interface).

As an open-source package, *WorldDynamics.jl* will also try to democratize access to distinct models as well as to promote transparency among them. Even if most of the current models are freely available for reproduction, they are usually implemented using proprietary software, which prevents us from precisely verifying their internal operation and, hence, how the models are simulated exactly. By using Julia and its notable packages, *WorldDynamics.jl* will provide a flexible framework that allows the usage of several solvers and integration with different methods with a reduced effort. Its current features already include the implementation of the entire Club of Rome series of models with the possibility of easily replicating all the plots of their major variables that appeared in the literature and changing parameter values and systems of equations in order to evaluate different policies. In conclusion, *WorldDynamics.jl* will allow model construction in a simplified way while enabling the application of modern scientific computing techniques over new and classical models.

4.1.3 Real Time Simulator for Digital Twin and Hardware-In-Loop in the Electrical Power Networks Scenario

The added value of the work consists of the specific analysis of the contribution to the uncertainty of the RTS system and, in particular, of its virtual elements, which are typically neglected or not depending on when the simulations are performed. Furthermore, the characterization procedure for a calibrator, aimed at assessing the performance of the RTS and its virtual elements, has been described in detail.
Moreover, the features of a stand-alone merging unit (SAMU) can be implemented and tested in a hardware-in-the-loop (HIL) environment, allowing us to simulate a complete digital substation inside the laboratory. What is presented in the following allows us to exploit existing technologies of a laboratory environment for the development of new equipment with additional functionalities.

4.1.4 Interactive Computing Service

The IAC framework is currently developed with a Hybrid Cloud HPC structure. This approach can be generalized to a broader range of architectural design, with any kind of host (e.g., bare metal nodes, purely VM-based environment, and so on) behind the scenes. For the purpose of CN-HPC the plan is to port such a framework on the Leonardo supercomputer, with further enhancements by evaluating the chance of high availability and load balancing services. Moreover, both in the backend and frontend side, additional improvements will be implemented. On the back-end side, the next step will be to avoid the intermediate dedicated Slurm controller, using the production scheduler of the cluster: this will largely simplify the design as well as authentication procedures and accounting. On the front-end side, the interface can be expanded by adding several features beyond the current status. The plan is to add additional tools to the front-end launchers, such as a web-based Virtual Network Computing (VNC) interface. Another feature that will be implemented, which clearly shows the potential of this kind of implementation, is the possibility to monitor in real-time the usage of resources while the computation is running; this is a particularly useful feature when GPUs are employed (a common case for AI applications), so that an on-the-fly visualization of the GPU occupancy can give a rough picture of possible bottlenecks in the workflow. To address the need to access an S3 bucket, some tests will be performed with dedicated tools that provide a quick interface to an existing object-storage system. In order to put in production the IAC platform, a new host will be configured by applying all the needed security measures and by integrating it properly with the already existing HPC production environment.

4.1.5 Jupyter Workflow

Jupyter Workflow demonstrates a novel methodology able to unleash Jupyter Notebook's superior productivity and portability features in the HPC area, explicitly targeting Cloud resources and their workload managers, HPC platforms with their system software, and their coupled exploitation usage. In this view, Jupyter Workflow is the first representative of a new class of interfaces for modern HPC applications. Jupyter Workflow has been successfully tested on several application pipelines from diverse scientific domains (e.g., molecular dynamics simulation, deep learning, bioinformatics, geoscience) and running on different combinations of Cloud and HPC facilities. However, enriching the expressive power of the literate workflow model with new control-flow constructs (e.g., conditional and iterative executions) and optimization rules (e.g., efficient reduce-by-key semantics or streaming

communications between workflow steps) is a crucial improvement to model and orchestrate a broader class of HPC workflows.

4.1.6 StreamFlow

The heterogeneity and complexity of modern applications force monolithic approaches to give way to modular architectures and patterns for designing and developing software, of which workflow models are first-class representatives. In the same way, heterogeneity in contemporary hardware resources (e.g., highly parallel hardware accelerators, low energy-consuming FPGAs, or application-specific quantum solvers) fosters modular approaches in designing execution environments for such applications. Hybrid workflows represent an essential methodological step in this direction. An explicit representation of the entire environment generalizes the concepts of portability and reproducibility from the application plane to the entire execution process. The separation of concerns brought by hybrid workflows promotes cooperation between domain experts, who write the application logic, and computer scientists, who find the best execution environment for each workflow step according to specific requirements (e.g., in terms of cost, time-to-solution or energy consumption). At the same time, both of them are free from the burdens of managing applications deployment and life-cycle and writing explicit data transfer logic, enhancing productivity. Finally, a loose mapping relation between steps and locations allows for automatic cross-stack executions of independent steps, providing a trivial way to offload tasks in urgent computing scenarios. Being tied to the CWL standard, StreamFlow's expressive power is not yet ready to easily design all the typical HPC workflows. Indeed, execution patterns like iterations, co-scheduling, and even co-location of multiple workflow steps are first-class citizens in HPC workflows, but they are not yet supported in CWL. A community effort to extend CWL with these workflow constructs is currently ongoing.

4.1.7 Parallel Multi-density Clustering

The CHD algorithm shows encouraging clustering accuracy on state-of-art datasets. However, it exhibits some limitations from the computational viewpoint when it is applied to a large volume of data. In fact, the neighborhood density computation and the multiple density-based clustering executions are very high time-consuming steps. Also, during some preliminary tests, it has been noticed that its sequential implementation is not feasible to analyze significant real-world datasets, whose high complexity and large volumes require more scalable solutions. For such a reason, research studies must be focused on the design, implementation, and evaluation of a parallel solution of CHD, whose execution on HPC and large-scale computing infrastructures could give several benefits in terms of both execution time, speed-up and scale-up.
Currently, a sequential implementation of CHD is available in Python. In order to implement a parallel solution of CHD, the computational granularity of each step should be analyzed, identify the bottlenecks and re-design the steps whose execution can be parallelized. Finally, the experimental evaluation must be performed on a HPC system, to assess the computational scalability of the proposed solution.

4.1.8 aMLLibrary

aMLLibrary has several perks which are specifically tailored for the building of performance models. It includes plugins for several common data pre-processing techniques, such as data normalization and one-hot encoding for discrete features, as well as other convenient tools such as row selection and data validity checks. It also supports automatic feature engineering, in the form of computation of logarithms, inverse values, and feature products/polynomial expansion up to a given degree. These tools can be useful to unearth

potentially relevant information hidden in the input features, such as quadratic dependencies and interaction terms. Feature selection techniques are also supported, including forward Sequential Feature Selection (SFS) [148] and importance weight selection by using the XGBoost regression model.
The main strengths of aMLLibrary with respect to other state-of-the-art libraries are its ease of use, customizability, and extensibility. A simple configuration text file is required to launch a full experimental campaign for all implemented models. This input file is declarative, therefore a collection of ML models can be used without writing a single line of Python code. Default probability distributions for hyperparameters are provided, and they are general enough to allow the automatic tuning mechanism to find the appropriate parameter values without further input by the user, which is most useful for those inexperienced with ML. At the same time, the user has full control over the experimental campaign thanks to the many options and flags available in the configuration file. Finally, extensibility is a major advantage for advanced users who wish to implement new data pre-processing techniques or new regression models in the aMLLibrary environment. One can simply write a plugin or a model wrapper and add it to the library, while exploiting or building on the existing features already available.
Individual analyses conducted by aMLLibrary can compare in a single run multiple alternative ML methods, and parallel processing of the training phase of the models is supported, to exploit the potentially large computational power available on the machine to which it is deployed. The user can specify the number of parallel cores to be used, and the library automatically distributes the training experiments evenly among the parallel workers, even if the underlying scikit-learn model is limited to single-thread execution. Furthermore, the library implements a fault tolerance mechanism by saving incremental progress checkpoints. If the experimental campaign is interrupted, e.g., because of the failure of the server the library is running on, it can recover the previous results and resume from there.

4.2 Static optimization and transformation layer

4.2.1 Sieve, Process, and Forward (SPF)

The adoption of purposely designed information and service models based on Value-of-Information (VoI) methodologies and tools presents significant advantages for the realization of self-adaptive and composition-friendly services operating in the Compute Continuum. More specifically, the SPF information-centric and value-based service model enables the streamlined development and management of services that can be either in edge devices, i.e., in the fog, or in the cloud, migrate to different computing platforms, and automatically scale their computation and bandwidth requirements according to the current execution context. The approaches adopted by SPF to explore interesting trade-offs between information processing speed and accuracy, leveraging VoI based prioritization and content-based filtering, have proved to be very effective and capable of bringing important advantages not only for fog services running on edge devices but also for those running in the Cloud.
To better support the deployment of HPC applications, it would be interesting to extend the VoI model defined within SPF for considering the peculiarities of these applications. Specifically, defining a formal notation to quantify the utility that HPC applications can bring to the general public could help SPF to prioritize the execution of computing-intensive HPC applications over general-purpose fog services when needed.

4.2.2 ParSoDA: Parallel Library for Big Data Analysis

ParSoDA was designed to facilitate the development of complex data analysis applications, mainly because it provides a set of data collection, filtering, transformation, analysis functions and algorithms that are typically used in the context of social and web data analysis. However, users are free to extend these functions with their own if they need any custom behavior. ParSoDA defines a quite generic scalable and distributed execution flow, which can support any type of data analysis application. In addition, it is characterized by a modular and extensible structure that can be easily extended to support other application contexts and/or execution runtimes. To make ParSoDA more usable in large-scale HPC systems, it could be extended to support integration with other runtime systems, including support to more scalable distributed storage and/or data structures.

4.2.3 BLEST-ML (BLocksize ESTimation via Machine Learning)

BLEST-ML introduces several contributions in the block size estimation task, with respect to the state-of-the-art of static and dynamic techniques for data partitioning and automatic tools used in HPC contexts for hyper-parameter tuning.
With respect to static approaches, which define a priori how data should be partitioned, our approach takes into consideration dataset characteristics and algorithm features. In addition, it can determine a suitable partitioning before the execution, using a machine learning model. In this way, it avoids the overhead introduced by dynamic approaches, which adapt data partitioning to the actual workload at runtime by dynamic adjustments. Finally, machine learning-based solutions like BLEST-ML can be more accurate and faster than general-purpose autotuners, which perform a resource-intensive exploration of vast search spaces, also adapting at runtime through dynamic profiling. Nonetheless, machine learning-based approaches may require a large amount of historical training data in order to generalize well to new or unseen inputs and are generally tailored to a specific optimization problem. To improve BLEST-ML and to make it more usable in a large-scale HPC context, the plan is to extend it to support the choice of other parameters required to configure a distributed environment, such as the number of nodes and the RAM to be assigned to each node. In addition, the methodology could also be applied to other frameworks and libraries for HPC systems other than PyCOMPSs and dislib, as it could be exploited in any case where data partitioning is essential to improve application performance and scalability.

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

At the best of our knowledge, no open-source software is available for compressing large collections of textual and source-code files. As an example, the Software Heritage archive is relying on the compression of individual files by the classic gzip.
Currently a prototype of the PPC instantiation that hinges on a single-threaded implementation is available, and it can manage GBs of data, but it is not able to scale to TBs/PTs.
The aim is at packaging an improved PPC instantiation that can work in a parallel and distributed scenario by adopting models, frameworks, and tools for parallel and distributed batch processing across clusters. Moreover, the plan is to systematically experiment the SimHash, MinHash and Winnowing fingerprints, as well as implement and test other permuting and partitioning strategies.

4.2.5 MALAGA, MultidimensionAL Big DAta Analytics over Massive Graph DAta

MALAGA introduces the proposal of a comprehensive framework for supporting big multidimensional data analytics over massive graph data, thus defining a plethora of emerging open problems that are related to both the implementation and the optimization

of the various big multidimensional data analytics tasks. These open problems can be addressed by defining innovative techniques and algorithms that take into consideration the well-understood properties of big data, including volume, velocity and variety, which become even worse when associated with graph-shaped data sources. These techniques and algorithms are the most important progress of the MALAGA framework, which will be considered in the PNRR CN-HPC research project. Last but not least, defining and implementing suitable case studies showing the capabilities of MALAGA in emerging big graph data applications and systems is another goal to be pursued by the project activities. On the other hand, MALAGA must be prone to be used in a high-performance computing setting, according to the project's guidelines. In order to achieve this nice amenity, it is necessary, for a side, to integrate MALAGA with state-of-the-art platforms for big data processing, like Hadoop, and big data management/analytics, like Spark, and, from another side, to design and develop specialized (optimized) solutions for big multidimensional data representation, management and indexing in distributed environments.

4.2.6 FastFlow/WindFlow: high-level and efficient streaming

One of the highest priorities nowadays in the context of software ecosystems is to provide programmers with practical and effective parallel programming tools targeting single high-end servers and many interconnected server platforms with the same level of offered performance. Additionally, there exists a need for programmers to use tools designed to coordinate and connect heterogeneous parts of an extensive distributed application without rewriting parts of the application when migrating from a classical HPC infrastructure to a more heterogeneous Cloud/Edge distributed platform. The FastFlow parallel programming framework aims to contribute to this research direction. The objective is to provide a run-time system capable of squeezing the maximum performance of the underlying single-node platform and being able to overlap computation and communication to amortize distributed communications yet enabling an easy transition from pure shared-memory applications to mixed shared- and distributed-memory applications without the need to mix multiple programming models. The streaming data-flow paradigm and the Building Blocks concepts offered by the FastFlow programming represent a solid feature not always present in other well-established frameworks. Additionally, with the evolution of single-node hardware, which comprises advanced heterogeneous multicores and accelerators such as GPUs and FPGAs, traditional scale-out streaming engines have proven ineffective in exploiting such new architectural facilities [149]. The WindFlow library tries to fill this gap by providing a scale-in stream processing library capable of offloading streaming computations on hybrid hardware while providing a high-level programming approach.

4.2.7 Clustering Algorithm

Before effectively deploying networks with distributed decision points, a number of crucial concerns must be tackled. This includes figuring out strategies for balancing and task scheduling, devising real-time data streaming protocols, ensuring the fault tolerance of distributed applications, and creating energy-efficient mixed precision algorithms. Further investigation and experimentation in actual use cases will be necessary to address these challenges and progress forward, drawing on techniques and approaches developed for similar high-performance distributed systems [150].

4.2.8 High performance and Low Power Hyperspectral Image Analysis

Preliminary results with synthetic dataset encourage further investigation of the problem, but before we can effectively use the proposed method in real-world situations, it is necessary to conduct more research and experiments. For instance, it is necessary to analyze

real datasets that are not publicly available, exploit different data types, and utilize more efficient libraries for inference. Additionally, it is necessary to further analyze different energy modalities in the edge computing devices on other applications. Other important topics that need to be addressed are the management of communication networks in an autonomous vehicle and how to assess the reliability of the method while ensuring its security.

4.2.9 DivExplorer: Analyzing Machine Learning Model Behavior via Pattern Divergence

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. Moreover, DivExplorer allows the understanding of the model behavior at the subgroup and global levels. Furthermore, the proposed approach is model agnostic. Hence, it treats the classification model as a black box, without knowledge of its internal working.
The current version of DivExplorer has been successfully applied to understand ML model behavior on small- and large-scale data. The aim is to enhance it for understanding Big Data models. The efficient exploration of DivExplorer is suitable for parallel and distributed implementation, allowing its adoption in a Big Data context.

4.3 Orchestration layer

4.3.1 BookedSlurm

BookedSlurm takes a different approach to the Slurm job scheduler, extending its consolidated functionalities of resource management and job queues by allowing users to create resource reservations under a pay-per-use model using credits as a currency. The web calendar provides a graphic tool to examine the cluster state and to select resources to reserve instead of relying only on CLI tools. Jobs billing values get a more profound meaning when associated with an actual currency cost, allowing one to differentiate between a job submitted in the queue and a job with a predefined reservation being scheduled before others in their time slot.

4.3.2 Orchestration of composite containerized applications in the Cloud continuum

Two simple principles guided the design of TORCH. The first principle concerns cloud users and their need to easily specify the requirements of the application to be deployed in the Cloud. We make the basic assumption that application owners are completely agnostic as to how their applications are handled on the Cloud. The second principle regards the application of a provisioning strategy that clearly separates the provisioning workflow from the actual invocation of cloud services that enforce the provisioning. Despite cloud providers expose proprietary APIs, the activities underlying any application provisioning process follow a common and API-independent pattern, that basically consists in instantiating the required resource(s) on the cloud provider infrastructure, deploying the application components on the instantiated resources, making the necessary software configuration, and finally running the application. By isolating the provisioning workflow from the actual cloud service invocation, several benefits derive in terms of reduced coding effort, enhanced maintainability, extensibility and scalability.
The INDIGO orchestrator offers a uniform interface to orchestrate computing resources and composite services in the cloud continuum, i.e., in the continuum of resources provided by

Cloud and Edge environments. The tool was also proved to work with GPU-powered resources for the purpose of running applications that require intense computation. Interaction with HPC-like environments was successfully tested via ad-hoc connectors which are far from being intuitive and in a form that can be proposed for a large adoption. In that regard, the tool must be enhanced to fully support the orchestration and management of HPC clusters.

4.3.3 Liqo

Different from current approaches, his project proposes a novel architectural paradigm: liquid computing, which builds upon and extends the well-established cloud and edge computing approaches towards an endless computing continuum. Then, we present a first real implementation of a software framework enabling a continuum of computational resources and ready-to-consume services spanning across multiple physical infrastructures. Overall, the resulting computing domain abstracts away the specificity of each cluster, presenting to the final users, either actively participating as actors or simply renting off-the-shelf services, a unique and borderless pool of available resources, the so-called big cluster. Thanks to this abstraction, applications are no longer constrained in a specific silo, but free to fly in the entire infrastructure, selecting the most appropriate location depending on its requirements (e.g., a user facing service may be replicated at the edge to account for low latency, while another might be constrained to European infrastructures to comply with GDPR), and the available resources, while retaining full compatibility (hence, models, tools, and commands) with vanilla Kubernetes.

4.3.4 Energy efficient orchestration and resource management in the cloud continuum

The current centralized architecture that characterizes current cloud computing infrastructures confines resources into data centers far from users and cyber-physical systems. This architecture does not support the development of applications and services that are characterized by stringent QoS requirements. Critical applications, e.g., industrial, might require low latency and high reliability communication with cyber physical systems or users, which is not feasible when the application logic is deployed on datacenters that communicate over long-distance links with significant delay and prone to failure. To overcome this limitation a cloud-continuum architecture [151] is envisioned as the future development of cloud computing. The cloud continuum includes an architecture where computing and storage resources are also installed in the intermediate computing layers between the data centers and the edge of the network to support the execution of applications and services that can benefit from low latency and reliable communication with users and cyber-physical systems. In this context, our proposed approach for orchestration and resource management and many of the proposed approaches from the literature are not adequate as they are designed specifically for a centralized computing environment installed in a single location. A revisitation of the current proposed approach is needed to manage a distributed computing infrastructure with heterogeneous computing nodes and network connections, considering also energy efficiency. Other resource management solutions have been recently proposed in literature, e.g. [152], [153], however, a holistic approach is still missing where network and computing resources are managed jointly with also considering energy efficiency aspects at large.

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

While Serverledge provides a suitable framework for low-latency FaaS execution in the Edge-Cloud continuum, several challenges still need to be addressed in order to fully support QoS-aware execution and scheduling in such a dynamic environment.
Starting with the support of complex FaaS applications, Serverledge can be extended to provide function composition by orchestrating the execution of multiple functions organized in an application workflow. The support of complex applications also requires the design of strategies and mechanisms to effectively manage not only stateless but also stateful FaaS functions.
While offloading provides a useful mechanism to redirect the function execution to another node, migration allows to move seamlessly a running function from the current node to another one. Migration support can be particularly helpful for long-running functions that characterize serverless data analytics and machine learning applications at the edge. Function offloading and migration mechanisms call for the design of offloading and migration policies that can drive the corresponding decision. As a global formulation of the decision problem for the whole system would not scale for realistic deployments, we envision the development of decentralized migration and offloading policies. Furthermore, their design can be integrated in a holistic way, also taking into account how functions are orchestrated within the application workflow and whether they use state.
As regards lightweight virtualization techniques, Serverledge currently employs a common solution to execute functions, which relies on running functions within Docker containers. Other containerization engines as well as microVMs (e.g., Amazon Firecracker), that inherit the strong isolation of VMs with the minimal management overhead of containers, can be integrated within Serverledge. The integration of lighter function sandboxing techniques, such as WebAssembly, is another direction to pursue in order to reduce the cold start issue and increase the number of functions deployed onto the same node thanks to the reduced memory footprint.
The deployment of Serverledge on the underlying computing nodes is currently static but it can be adapted at runtime, for example taking into account node heterogeneity. The nodes can be selected through properly designed placement policies as well as scaled by means of auto-scaling reinforcement-based policies [154]. The design of coordinated cross-layer policies that span from computing nodes, which are geographically distributed, to function execution environments deployed inside single nodes represents another challenging issue.
Finally, Serverledge can be extended towards a sustainable FaaS platform in the Cloud-Edge continuum [155]. Energy-efficient scheduling and offloading policies can be designed to effectively manage energy-constrained nodes (e.g., battery-powered sensors or smartphones) on which Serverledge can be deployed at the network edges. This requires an accurate understanding of where and how energy is consumed during function execution, so that functions can also be characterized by their energy footprint.

4.4 Runtime management layer

4.4.1 MoveQUIC: a QUIC-based toolbox for the live migration of microservices at the network edge

The existing toolbox has two main limitations. First, the current version has been successfully tested in small-scale lab environments, focusing on the performance from the perspective of a single user/single service. To better fit the HPC context, a focus should be put into the ability of the framework to scale without impacting the service performance, possibly supporting the deployment in a distributed computation environment. Second, being a prototype implementation, the existing framework is not ready for production deployments.
Regarding the first limitation, the toolbox should be modified from a standalone object, which is not integrated with existing container-orchestration tools, to a more holistic

solution, relying on standardized solutions or based on de-facto standards such as Kubernetes. Providing such an integration would, on the one hand, enable the stateful migration of multiple containers (or better, of Pods) either within the same cluster or among multiple distributed clusters and, on the other hand, improving the application of the toolbox itself at scale, e.g., applying the migration techniques to load-balancing scenarios, or enabling a quick (re)start of complex services, avoiding the so-called “cold start” problem. Moreover, although our tool introduces migration times that are in line with those available in the literature, the service downtimes introduced by stateful-migration process can still impair the performance of the service itself, especially in cases wherein the request rate is high. To cope with these aspects the container migration procedure needs to be optimized, possibly taking into account knowledge on the application behavior. Finally, as highlighted in the description of CRIU, transferring container status from the source to the target computing node is a crucial task in the migration procedure. In this respect, the toolbox could be extended to support several mechanisms for transferring the container status, e.g., relying on a third node acting as an external repository, or relying on a distributed file system, etc.
Regarding the second limitation, instead, the elements of the toolbox would need to be implemented within production-grade software (e.g., Meta’s implementation of QUIC) and/or deployed on a real edge platform, to be tested at scale. In this respect, integrating the toolbox within a fully-fledged orchestration platform would also foster the application of our toolbox into production environments.

4.4.2 Nethuns: a library for efficient, socket-independent network I/O

Most Cloud-based data centers currently use DPDK-based solutions to accelerate access to network I/O. However, DPDK has been left out of the first implementation of nethuns since its callback-based programming model differs in fundamental ways from the other engines. To increase the number of high-performance networking applications that can benefit from the ease of programmability and portability of nethuns, the framework must be extended to support the DPDK drivers.
Moreover, eBPF is emerging as a general tool for kernel programmability in the Linux world, with important hooks in the network stack, where downloaded eBPF programs can implement custom packet processing and traffic control. High performance applications can leverage eBPF to reduce the number of kernel/userspace context-switches and by lowering the processing latency of network data. The current implementation of nethuns, however, is compatible with eBPF only when the underlying engine is AF_XDP, and even in that case the user interface is cumbersome. Progress in this area is needed, to design a network programming API that understands eBPF programs independently of the engine, possibly integrating techniques like eBPF in userspace.

4.4.3 CAPIO: cross-application programmable I/O

CAPIO proposes a novel methodology to transparently inject I/O streaming capabilities into scientific workflows, improving the I/O performance without modifying the applications. The novelty of this work is twofold: the I/O coordination language allows users to express workflow data dependencies with streaming semantics, while the CAPIO runtime transforms a batch execution into a streaming execution following the semantics given using the coordination language.

4.4.4 INSANE: A Uniform API for QoS-aware Host Networking as a Service

INSANE significantly simplifies the development of latency-critical services based on acceleration techniques by offering QoS-aware host networking as a service, introducing

minimal (ns-scale) overhead, and making applications portable across heterogeneous environments such as public clouds and edge cloud nodes, thus unleashing a new generation of ultra-low latency cloud continuum applications in several time-critical domains. For HPC applications, libfabric is currently the most advanced and widely used communication API. Also known as Open Fabrics Interfaces (OFI), it defines a communication API for high-performance parallel and distributed applications, and is a low-level communication library that abstracts several networking technologies. However, INSANE is designed as an high-level alternative to libfabric, particularly for the future of HPC, where Ethernet networks are expected to replace proprietary technologies like Infiniband. Moreover, INSANE's architecture makes it easy to integrate additional technologies, including those used in HPC, storage access, and GPUs.

4.5 Hardware layer

4.5.1 MLIR: multi-level intermediate representations for heterogeneous computing platform

MLIR can improve the current state-of-the-art thanks to its design philosophy, mainly:

- Domain-specific optimization: MLIR supports domain-specific optimizations by design, which can lead to better performance and more efficient code generation. This feature is essential in ML and other scientific computing domains, where specialized algorithms and data structures are often used.
- Flexible representation: MLIR provides a flexible representation that is customizable to meet the needs of different domains and applications. This approach makes developing tools and compilers tailored to specific use cases faster and easier to maintain and evolve.
- Modularity: MLIR has a modular design, making it easier to develop and maintain complex systems and integrate them with existing tools and frameworks.

Eventually, MLIR provides a powerful and flexible approach to developing compilers and other tools for ML and other domains. Its domain-specific optimizations and flexible representation make it a compelling choice for developers looking to improve performance and efficiency in their applications. Up to now, MLIR has been adopted in a few application domains, while the potential benefits mentioned above can be extended to more HPC areas. In this context, the proposed tool aims to bridge this gap.

5 Synergies between Use Cases and Tools

Some of the tools presented in the previous Sections can be exploited synergistically to cope with some typical use cases for HPC applications. In this Section, we illustrate some preliminary relevant use cases identified by the Flagship partners that require the adoption of HPC tools and that provide a preliminary basis for the investigation of the industrial prototypes that will be involved in the Flagship's future activities.
These use cases include the optimization of complex astrophysical processes and the processing of large-scale genomic data. Moreover, some of the partner's HPC tools are already adopted in existing prototypes that can benefit from integration with other HPC tools. By leveraging our presented HPC tools, these use cases can be addressed more efficiently and accurately, leading to significant improvements in research and innovation.

The synergies between use cases and tools will be fully presented and analyzed in the second deliverable, D5.FL3: Selection of candidate prototypes for frameworks and development tools for HPC.

- *Astrophysics data analysis and visualization*
  - Over the years, the astrophysics domain has developed a set of ad-hoc tools and software modules to tackle the challenging particularities of the field. 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

- *Modelling galaxy formation and clustering*
  - Theoretical astrophysics heavily exploits advanced numerical algorithms and High Performance computational tools to model the highly nonlinear phenomena involved with stellar formations and evolution within galaxies.
  - The environment of a galaxy is a highly nonlinear environment, hosting stellar populations which form from the interstellar gas in a process driven by gravitational instability and strongly affected by radiation heating and cooling. In turn, stellar radiation can significantly affect the ability of the interstellar medium to form new stars. These *feedback mechanisms* operate on comparable spatial and temporal scales. The formation of *Supermassive Black Holes* (hereafter SMBH) at the centers of galaxies adds a further feedback component.

- *Genomic variant calling pipeline*
  - Genomic analysis is becoming more and more a routine activity in several hospitals and research labs. Genomic data coming from Next Generation Sequencing (NGS) devices requires the definition of a dedicated multi-step pipeline for variants identification (i.e., variants calling), annotation and prioritization.

Some of the presented tools that are ready to be exploited, instead, are summarized in the following:

- *Interactive Computing Service*
  - The IAC framework 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, 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.
  - This general approach allows to test different combinations of frontends and backends exploiting novel configurations in the context of the CN-HPC while deploying the tools listed in Section 2 of this document.

  - With the same approach in mind, it will be possible to try to address the needs of all the partners of the CN-HPC. For instance, a valuable synergy and potential cooperation could be evaluated with the Jupyter Workflow tool and the StreamFlow tools.
- *VisIVO*
  - https://visivo.readthedocs.io/, [156] at the TRL7 level. Extended with the ViaLactea Visual Analytic [157] module (VLVA). 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.
  - Workflow abstractions to allow a portable representation of modular applications and their resource requirements, fostering reproducibility and maintainability should be investigated, to take advantage of heterogeneous HPC facilities (including also mixed HPC-Cloud resources) while minimizing data-movement overheads, exploiting the *StreamFlow* and the *Jupyter Workflow* presented in Section 2.
  - Additionally, fast I/O techniques could be exploited for optimizing importing of large-scale datasets (currently employing MPI). Such as: *CAPIO* and *Nethuns*.

- *Compression of peta-scale collections of textual and source-code files*
  - A single-threaded implementation of the UNIPI compression library is almost ready and has been successfully deployed to compress blobs of several dozens of gigabytes. New (single-threaded) algorithms are in preparation and will be designed and implemented in the next few months for achieving an even better compression ratio.
  - The parallel and distributed implementation is to be developed to achieve better time efficiency, given the significant size of the Software Heritage archive.
  - The compression libraries will be extended to a parallel and distributed scenario by adopting models, frameworks, and tools for parallel and distributed batch processing across clusters.
  - Possible synergies can be exploited with the Parallel Library for Big Data Analysis (*ParSoDA*) and *FastFlow/WindFlow*, and with the mirror of the Software Heritage archive set up by ENEA to perform the experiments needed for evaluating the efficiency and efficacy of the tool.

- *Liqo.io*
  - Cloud/HPC Federation. Several real problems advocate the usage of multiple computing infrastructures at the same time. Some examples are (a) job bursting (e.g., when computing jobs are so intense to consume all the resources in a given clusters, hence benefiting from the possibility of borrowing spare resources in other clusters), and (b) data gravity-driven computing (e.g., when data, sparse across multiple infrastructure providers, require computing to be carried out on the different infrastructures, hence dispatching jobs where the data is located). This requires specific tools which are capable to enable seamless multicloud computing, hence enabling distributed computing to be carried out as everything would be in the same cluster.

- *WorldDynamics.jl*
  - A Julia framework for world dynamics modeling and simulation. It is a 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.
  - A first version of the package is available at https://github.com/worlddynamics/WorldDynamics.jl. It includes the implementation of the entire WorldX series of models of the Club of Rome.

- *Genomic variant calling pipeline*

  - Several intermediate tools are used and workflow managers are employed to orchestrating such pipelines since it is crucial to obtain an efficient and manageable execution. A pipeline has been defined and implemented, which is currently in production (via Nextflow) and that takes advantage of GPUs computing power through the Parabricks (https://docs.nvidia.com/clara/parabricks/4.0.0/softwareoverview.html) tools suite.
  - The current implementation of the pipeline should be adapted to the Streamflow environment with the aim of increasing the flexibility and minimizing data movements. Streamflow and its Jupyter notebook implementation will be used. The increased flexibility will allow to test the pipeline in several other (possibly heterogeneous) execution.

- *ParSoDA*
  - Parallel Library for Big Data Analysis (ParSoDA) facilitates the development of complex data analysis applications, supporting, in its initial version, two of most leading parallel and distributed computing frameworks, i.e., Hadoop and Spark.
  - The main contributions of ParSoDA can be summarized: i) to define a general structure for data analysis applications; ii) provide an extensible set of functions that can be used and combined during the different steps of the computation; iii) reduce the programming skills needed for implementing scalable data analysis applications; iv) reduce the execution time of data analysis by parallelizing the execution of the code and exploiting the computational and storage resources of clusters.
  - The library was initially intended to facilitate the processing and analysis of data from social media. In fact, it natively includes data collectors and parsers to handle data from the main social media (Facebook, Twitter, Flickr). However, the library is easily extendable to support other data sources.
  - The code of ParSoDA is available as open-source software at https://github.com/SCAlabUnical/ParSoDA.

- *BLEST-ML*
  - BLocksize ESTimation via Machine Learning (BLEST-ML) is designed to find a suitable estimate for the block size to be used to run data-parallel applications in distributed HPC environments.
  - The current version of BLEST-ML does not make use of parallel programming. Parallelism could be exploited in a future version of BLEST-ML both for the training of the machine learning model and for generating the execution logs, in the case they are not available for the training phase.
  - A first implementation of BLEST-ML, tailored to the dislib library for distributed computing, is available at the following link: https://github.com/eflows4hpc/dislib-block-size-estimation

- *DivExplorer*
  - DivExplorer is an anomalous subgroup identification approach to automatically identify data subgroups for which a classification model performs differently than overall behavior. Data subgroups are identified by slicing a tabular dataset in the attribute domain. The approach integrates frequent pattern mining techniques to extract subgroups efficiently with adequate representation in the data.
  - The first version of DivExplorer is available as an open-source Python package at https://github.com/elianap/divexplorer.git and on the Python Package Index (PyPI) repository https://pypi.org/project/divexplorer/.

6 References

[1] Apache Software Foundation, "Apache Kafka project," [Online]. Available: https://kafka.apache.org/. [Accessed 05 04 2023].

[2] Apache Software Foundation, "Apache Flink framework," [Online]. Available: https://flink.apache.org/. [Accessed 05 04 2023].

[3] Apache Software Foundation, "Apache Storm clustering framework," [Online]. Available: https://storm.apache.org/. [Accessed 05 04 2023].

[4] G. Mencagli, P. Dazzi and N. Tonci, "SpinStreams: a static optimization tool for data stream processing applications," in *Proceedings of the 19th International Middleware Conference*, 2018.

[5] Canonical Ltd, "Linux containers," [Online]. Available: https://linuxcontainers.org/. [Accessed 28 03 2023].

[6] Docker Inc, "Docker," [Online]. Available: https://www.docker.com/. [Accessed 06 04 2023].

[7] OASIS, "TOSCA Simple Profile in YAML Version 1.3," [Online]. Available: http://docs.oasis-open.org/tosca/TOSCA-Simple-Profile-YAML/v1.3.

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

[9] P. Crescenzi, E. Natale and P. B. Serafim, "WorldDynamics.jl: v0.1.0," 2022. [Online]. Available: https://github.com/worlddynamics/WorldDynamics.jl,. [Accessed 24 03 2023].

[10] D. H. Meadows, D. L. Meadows, J. Randers and W. W. Behrens III, The Limits to Growth; A Report for the Club of Rome's Project on the Predicament of Mankind, New York: Universe Books, 1972.

[11] A. Mingotti, F. Costa, L. Peretto and R. Tinarelli, "Characterization procedure for stand-alone merging units based on hardware-in-the-loop technology," *Energies,* vol. 14, no. 3, April 2021.

[12] C. Allioua, F. Costa, A. Mingotti, L. Peretto and R. Tinarelli, "Characterization procedure for stand-alone merging units based on hardware-in-the-loop technology," in *Proc. of the 2022 12th IEEE International Workshop on Applied Measurements for Power Systems*, 2022.

[13] C. Allioua, F. Costa, A. Mingotti, L. Peretto and R. Tinarelli, "Open-Source MATLAB-Based PMU Library for HIL Applications Compliant with IEC/IEEE 60255-118-1," in *12th IEEE International Workshop on Applied Measurements for Power Systems*, 2022.

[14] A. Mingotti, F. Costa, D. Cavaliere, L. Peretto and R. Tinarelli, "On the importance of characterizing virtual pmus for hardware-in-the-loop and digital twin applications," *Sensors,* vol. 21, no. 18, 2021.

[15] A. Mingotti, F. Costa, L. Peretto and R. Tinarelli, "Characterization procedure for stand-alone merging units based on hardware-in-the-loop technology," *Energies,* vol. 14, no. 7, 2021.

[16] CINECA, "UG3.5: ADA Cloud UserGuide," 2022 UG3.5. [Online]. Available: https://wiki.u-gov.it/confluence/display/SCAIUS/UG3.5%3A+ADA+Cloud+UserGuide.

[17] CINECA, "UG3.3: GALILEO100 UserGuide," 2022 UG3.3. [Online]. Available: https://wiki.u-gov.it/confluence/display/SCAIUS/UG3.3%3A+GALILEO100+UserGuide.

[18] I. Colonnelli, M. Aldinucci, B. Cantalupo, L. Padovani, S. Rabellino, C. Spampinato, R. Morelli, R. Di Carlo, N. Magini and C. Cavazzoni, "Distributed workflows with Jupyter," *Future Generation Computer Systems,* vol. 128, pp. 282-298, 2022.

[19] I. Colonnelli, B. Cantalupo, I. Merelli and M. Aldinucci, "StreamFlow: cross-breeding cloud with HPC," *IEEE Transactions on Emerging Topics in Computing,* vol. 9, no. 4, pp. 1723-1737, 2021.

[20] M. R. Crusoe, S. Abeln, A. Iosup, P. Amstutz, J. Chilton, N. Tijanic, H. Ménager, S. Soiland-Reyes, B. Gavrilovic, C. A. Goble and The CWL Community, "Methods included: standardizing computational reuse and portability with the Common Workflow Language," *Communications of the ACM,* vol. 65, no. 6, pp. 54-63, 2022.

[21] E. Cesario, P. I. Uchubilo, A. Vinci and X. Zhu, "Multi-density Urban Hotspots Detection in Smart Cities: A Data-driven Approach and Experiments," *Pervasive and Mobile Computing,* vol. 86, pp. 1-13, 2022.

[22] Draper and Smith, "Applied Regression Analysis," *Wiley,* 1998.

[23] Chen and Guestrin, "XGBoost: A Scalable Tree Boosting System," in *SIGKDD*, 2016.

[24] 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.

[25] J. Conejero, S. Corella, R. M. Badia and J. Labarta, "Task-based programming in COMPSs to converge from HPC to big data," *The International Journal of High Performance Computing Applications,* vol. 32, no. 1, pp. 45-60, 2018.

[26] B. S. Center, "Pycompss distributed data set user manual," 2018. [Online]. Available: http: //compss.bsc.es/releases/compss/latest/docs/DDS_Manual.pdf.

[27] R. Cantini, F. Marozzo, A. Orsino, D. Talia, P. Trunfio, R. M. Badia, J. Ejarque and F. Vázquez, "Block size estimation for data partitioning in HPC applications using machine learning techniques," *arXiv preprint arXiv:2211.10819,* 2022.

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

[29] C. Chen, X. Yan, F. Zhu, J. Han and P. S. Yu, "Graph OLAP: A Multi-Dimensional Framework for Graph Data Analysis," *Knowl. Inf. Syst,* vol. 21, no. 1, pp. 41-63, 2009.

[30] M. Aldinucci, M. Danelutto, P. Kilpatrick and M. Torquati, "Fastflow: high-level and efficient streaming on multi-core," *Programming multi-core and many-core computing systems, parallel and distributed computing,* 2017.

[31] N. Tonci, M. Torquati, G. Mencagli and M. Danelutto, "Distributed-Memory FastFlow Building Blocks," *International Journal of Parallel Programming,* vol. 51, pp. 1-21, 2023.

[32] G. Mencagli, M. Torquati, A. Cardaci, A. Fais, L. Rinaldi and M. Danelutto, "WindFlow: High-Speed Continuous Stream Processing With Parallel Building Blocks," *IEEE Transactions on Parallel and Distributed Systems,* vol. 32, no. 11, pp. 2748-2763, 2021.

[33] M. Lapegna, W. Balzano, N. Meyer and D. Romano, "Clustering Algorithms on Low-Power and High-Performance Devices for Edge Computing Environments," *Sensors,* vol. 21, 2021.

[34] G. Laccetti, M. Lapegna and D. Romano, "A hybrid clustering algorithm for high-performance edge computing devices," in . *21st International Symposium on Parallel and Distributed Computing (ISPDC)*, 2022.

[35] 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," *Parallel Processing and Applied Mathematics,* vol. Lecture Notes in Computer Science , 2023.

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

[37] E. Pastor, A. Gavgavian, E. Baralis and L. de Alfaro, "How divergent is your data?," in *Proceedings of the VLDB Endowment*, 2021.

[38] O. Tomarchio, D. Calcaterra, G. Di Modica and P. Mazzaglia, "Torch: a tosca-based orchestrator of multi-cloud containerised applications," *Journal of Grid Computing,* vol. 19, no. 1, 2021.

[39] OMG, "Business Process Model and Notation," 2011. [Online]. Available: http://www.omg.org/spec/BPMN/2.0/ .

[40] A. Costantini, G. Di Modica, J. Ahouangonou, D. Duma, B. Martelli, M. Galletti, M. Antonacci, D. Nehls, P. Bellavista, C. Delamarre and D. Cesini, " IoTwins: Toward Implementation of Distributed Digital Twins in Industry 4.0 Settings," *Computers,* 2022.

[41] The INDIGO Orchestrator,, [Online]. Available: https://github.com/indigo-dc/orchestrator.

[42] Apache, "The MESOS project Mesos Project," [Online]. Available: http://mesos.apache.org/.

[43] "The Mesos-EDGE runtime," [Online]. Available: https://baltig.infn.it/IoTwins/mesos-edge.

[44] Apache, "The Marathon Framework," [Online]. Available: https://mesosphere.github.io/marathon/.

[45] Apache, "The Chronos Framework," [Online]. Available: https://mesos.github.io/chronos/.

[46] M. Catena and N. Tonellotto, "Energy-Efficient Query Processing in Web Search Engines," *IEEE Transactions on Knowledge and Data Engineering,* vol. 29, no. 7, 2017.

[47] G. Russo Russo, V. Cardellini and F. Lo Presti , "Serverless functions in the cloud-edge continuum: Challenges and opportunities," in *31st Euromicro International Conference on Parallel, Distributed, and Network-Based Processing*, 2023.

[48] G. Russo Russo, V. Cardellini, F. Lo Presti and T. Mannucci, "serverledge: Decentralized Function-as-a-Service for the edge-cloud continuum," in *21st International Conference on Pervasive Computing and Communications*, 2023.

[49] "Apache OpenWhisk," [Online]. Available: https://openwhisk.apache.org/. [Accessed 28 04 2023].

[50] S. Shillaker and P. Pietzuch, "faasm: Lightweight isolation for efficient stateful serverless computing," in *USENIX Annual Technical Conference*, 2020.

[51] P. Gadepalli, S. McBride, G. Peach, L. Cherkasova and G. Parmer, "sledge: A serverless-first, light-weight wasm runtime for the edge," in *21st International Middleware Conference*, 2020.

[52] T. Pfandzelter and D. Bermbach, "tinyfaas: A lightweight faas platform for edge environments," in *IEEE International Conference on Fog Computing*, 2020.

[53] 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.

[54] F. Barbarulo, C. Puliafito, A. Virdis and E. Mingozzi, "Extending ETSI MEC Towards Stateful Application Relocation Based on Container Migration," in *IEEE 23nd International Symposium on a World of Wireless, Mobile and Multimedia Networks*, 2022.

[55] N. Bonelli, F. Del Vigna, A. Fais, G. Lettieri and G. Procissi, "Programming socket-independent network functions with nethuns," *ACM SIGCOMM Computer Communication Review,* vol. 52, no. 2, pp. 35-48, 2022.

[56] M. Marty, M. de Kruijf, J. Adriaens, C. Alfeld, S. Bauer, C. Contavalli, M. Dalton, N. Dukkipati, W. Evans, S. Gribble, et al., "Snap: A microkernel approach to host networking," in *ACM Symposium on Operating Systems Principles*.

[57] I. Zhang, A. Raybuck, P. Patel, K. Olynyk, J. Nelson, O. Navarro Leija, A. Martinez, J. Liu, A. Kornfeld Simpson, S. Jayakar, et al., "The demikernel datapath os architecture for microsecond-scale data-center systems," in *Proceedings of the ACM SIGOPS 28th Symposium on Operating Systems Principles*, 2021.

[58] C. Lattner, M. Amini, U. Bondhugula, A. Cohen, A. Davis, J. Piennar, R. Riddle, T. Shpeisman, N. Vasilache and O. Zinenko, "MLIR: Scaling compiler infrastructure for domain specific computation," in *IEEE/ACM International Symposium on Code Generation and Optimization*, 2021.

[59] 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,* vol. 19, no. 4, 2022.

[60] A. Burrello, A. Garofalo, N. Bruschi, G. Tagliavini, D. Rossi and F. Conti, "Dory: Automatic end-to-end deployment of real-world dnns on low-cost iot mcus.," *IEEE Transactions on Computers,* vol. 70, no. 8, 2021.

[61] J. W. Forrester, World Dynamics, Cambridge, Massachusetts: Wright-Allen Press, 1971.

[62] S. Dixson-Declève, O. Gaffney, J. Ghosh, J. Randers and J. Rockström, Earth for All: A Survival Guide for Humanity, Per Espen Stoknes: New Society Pub, 2022.

[63] E4 Computer Engineering, 2023. [Online]. Available: https://www.e4company.com/en/gpu-appliance-for-artificial-intelligence/.

[64] Jupiter, 2023. [Online]. Available: https://jupyter.org.

[65] E. Deelman, T. Peterka, I. Altintas, C. D. Carothers, K. K. van Dam, K. Moreland, M. Parashar, L. Ramakrishnan, M. Taufer and J. S. Vetter, "The future of scientific workflows," *Int. J. High Perform. Comput. Appl.,* vol. 32, no. 1, p. 159–175, 2018.

[66] T. Kluyver, B. Ragan-Kelley, F. Pérez, B. E. Granger, M. Bussonnier, J. Frederic, K. Kelley, J. B. Hamrick, J. Grout, S. Corlay, P. Ivanov, D. Avila, S. Abdalla, C. Willing and e. al, "Jupyter notebooks - a publishing format for reproducible computational workflows," in *Positioning and Power in Academic Publishing: Players, Agents and Agendas, 20th International Conference on Electronic Publishing*, 2016.

[67] R. C. Thomas, S. Cholia, K. Mohror and J. M. Shalf, "Interactive supercomputing with Jupyter," *Comput. Sci. Eng,* vol. 23, no. 2, pp. 93-98, 2021.

[68] T. E. Odaka, A. Banihirwe, G. Eynard-Bontemps, A. Ponte, G. Maze, K. Paul, J. Baker and R. Abernathey, "The PANGEO ecosystem: Interactive computing tools for the geosciences: Benchmarking on HPC," *Tools and Techniques for High Performance Computing,* 2019.

[69] R. F. da Silva, R. Filgueira, I. Pietri, M. Jiang, R. Sakellariou and E. Deelman, "A characterization of workflow management systems for extreme-scale applications," *Future Generation Computer Systems,* vol. 75, p. 228–238, 2017.

[70] T. Fahringer, R. Prodan, R. Duan, J. Hofer, F. Nadeem, F. Nerieri, S. Podlipnig, J. Qin, M. Siddiqui, H. L. Truong, A. Villazón and M. Wieczorek, " ASKALON: A Development and Grid Computing Environment for Scientific Workflows," *Workflows for e-Science, Scientific Workflows for Grids,* 2007.

[71] E. Deelman, K. Vahi, G. Juve, M. Rynge, S. Callaghan, P. Maechling, R. Mayani, W. Chen, R. F. da Silva, M. Livny and R. K. Wenger, "Pegasus, a workflow management system for science automation," *Future Generation Comp. Syst,* vol. 46, p. 17–35, 2015.

[72] T. M. Oinn, R. M. Greenwood, M. Addis, M. N. Alpdemir, J. Ferris, K. Glover, C. A. Goble, A. Goderis, D. Hull, D. Marvin, P. Li, W. Lord, R. Pocock, M. Senger, R. Stevens, A. Wipat and C. Wroe, "Taverna: lessons in creating a workflow environment for the life sciences," *Concurrency and Computation: Practice and Experience,* vol. 18, no. 10, p. 1067–1100, 2006.

[73] I. J. Taylor, M. S. Shields, I. Wang and A. Harrison, "The Triana Workflow Environment: Architecture and Applications," *Workflows for e-Science, Scientific Workflows for Grids,* p. pp. 320–339, 2007.

[74] E. Afgan, D. Baker, M. van den Beek, D. J. Blankenberg, D. Bouvier, M. Cech, J. Chilton, D. Clements, N. Coraor, C. Eberhard, B. A. Grüning, A. Guerler, J. Hillman-Jackson, G. V. Kuster, E. Rasche, N. Soranzo, N. Turaga, J. Taylor and A. Nekrutenko, "The Galaxy platform for accessible, reproducible and collaborative biomedical analyses," *Nucleic Acids Research,* vol. 44, no. Webserver-Issue, W3–W10,, 2016.

[75] I. J. Taylor, M. S. Shields, I. Wang and O. F. Rana, "Triana Applications within Grid Computing and Peer to Peer Environments," *J. Grid Comput.,* vol. 1, no. 2, pp. 199-217, 2003.

[76] M. Siddiqui, A. Villazón, J. Hofer and T. Fahringer, "GLARE: A Grid Activity Registration, Deployment and Provisioning Framework," in *Proceedings of the ACM/IEEE SC2005 Conference on High Performance Networking and Computing*, 2005.

[77] D. Thain, T. Tannenbaum and M. Livny, "Distributed computing in practice: the Condor experience," *Concurrency - Practice and Experience,* vol. 7, no. 2-4, p. 323–356, 2005.

[78] J. A. Novella, P. E. Khoonsari, S. Herman, D. Whitenack, M. Capuccini, J. Burman, K. Kultima and O. Spjuth, "Container-based bioinformatics with Pachyderm," *Bioinformatics,* vol. 35, no. 5, pp. 1723-1737, 2019.

[79] "Cities: The century of the city," *Nature,* 2010.

[80] Izima, de Fréin and Malik, "A Survey of Machine Learning Techniques for Video Quality Prediction from Quality of Delivery Metrics," *Electronics 2021.*

[81] Maros, Murai, Couto da Silva, Almeida, Lattuada, Gianniti, Hosseini and Ardagna, "Machine Learning for Performance Prediction of Spark Cloud Applications," in *IEEE CLOUD*, 2019.

[82] Mustafa, Elghandour and Ismail, "A Machine Learning Approach for Predicting Execution Time of Spark Jobs," *Alexandria Engineering Journal,* 2018.

[83] Nawrocki and Osypanka, "Cloud Resource Demand Prediction using Machine Learning in the Context of QoS Parameters," *Journal of Grid Computing,* 2021.

[84] Lattuada, Gianniti, Ardagna and Zhang, "Performance Prediction of Deep Learning Applications Training in GPU as a Service Systems," *Cluster Computing,* 2022.

[85] Kirchoff, Xavier, Mastella and De Rose, "A preliminary study of machine learning workload prediction techniques for cloud applications," in *EuroPDP*, 2019.

[86] Das, Leaf, Varela and Patterson, "Skedulix: Hybrid Cloud Scheduling for Cost-Efficient Execution of Serverless Applications," in *IEEE CLOUD*, 2020.

[87] Ibrar, Wang, Muntean, Akbar, Shah and Malik, "PrePass-Flow: A Machine Learning based technique to minimize ACL policy violation due to links failure in hybrid SDN," *Computer Networks,* 2021.

[88] S. Amer-Yahia, N. Ibrahim, C. K. Kengne, F. Ulliana and M. C. Rousset, "Socle: Towards a framework for data preparation in social applications," *Ingénierie des Systèmes d'Information,* vol. 19, no. 3, pp. 49-72, 2014.

[89] Á. Cuesta, D. F. Barrero and M. D. R-Moreno, "A Framework for massive Twitter data extraction and analysis," *Malaysian J. of Computer Science,* vol. 27, no. 1, 2014.

[90] D. Zhou, L. Chen and Y. He, "An unsupervised framework of exploring events on twitter: Filtering, extraction and categorization," in *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence*, Austin, Texas, USA., 2015.

[91] A. Hussain and R. Vatrapu, "Social Data Analytics Tool (SODATO)," *Springer International Publishing, Cham,* pp. 368-372, 2014.

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

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

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

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

[96] 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.

[97] 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.

[98] 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.

[99] 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.

[100] D. Yan, Y. Bu, Y. Tian and A. Deshpande, "Big Graph Analytics Platforms," *Found. Trends Databases,* vol. 7, no. (1-2), pp. 1-195, 2017.

[101] D. Yan, Y. Tian and J. Cheng, "Systems for Big Graph Analytics," *Springer Briefs in Computer Science,* pp. 1-92, 2017.

[102] P. Sun, Y. Wen, T. Doung and X. Xiao, "GraphH: High Performance Big Graph Analytics in Small Clusters," *CLUSTER,* pp. 256-266, 2017.

[103] V. Papavasileiou, K. Yocum and A. Deutsch, "Online Provenance for Big Graph Analytics," in *SIGMOD Conference*, 2019.

[104] M. Hassani, P. Spaus, A. Cuzzocrea and T. Seidl, "I-HASTREAM: Density-Based Hierarchical Clustering of Big Data Streams and Its Application to Big Graph Analytics Tools," *CCGrid,* pp. 656-665, 2016.

[105] A. Cuzzocrea, "Innovative Paradigms for Supporting Privacy-Preserving Multidimensional Big Healthcare Data Management and Analytics: The Case of the EU H2020 QUALITOP Research Project," in *SWH@ISWC*, 2021.

[106] A. Cuzzocrea and P. G. Bringas, "CORE-BCD-mAI: A Composite Framework for Representing, Querying, and Analyzing Big Clinical Data by Means of Multidimensional AI Tools," in *HAIS*, 2022.

[107] A. Cuzzocrea, "Multidimensional Big Data Analytics over Big Web Knowledge Bases: Models, Issues, Research Trends, and a Reference Architecture," in *BigMM*, 2022.

[108] J. Gray, S. Chaudhuri, A. Bosworth, A. Layman, D. Reichart, M. Venkatrao, F. Pellow and H. Pirahesh, "Data Cube: A Relational Aggregation Operator Generalizing Group-by, Cross-Tab, and Sub Totals," *Data Min. Knowl. Discov.,* vol. 1, no. 1, pp. 29-53, 1997.

[109] L. Golab and M. Tamer Özsu, "Issues in data stream management," in *SIGMOD*, 2003.

[110] H. Grahn and P. Geladi, "Techniques and applications of hyperspectral image analysis," *John Wiley & Sons,* 2007.

[111] D. Baylor, 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.

[112] M. Kahng, D. Fang and D. H. Chau, " Visual exploration of machine learning results using data cube analysis," in *Proceedings of the Workshop on Human-In-the-Loop Data Analytics*, 2016.

[113] Á. A. Cabrera, W. Epperson, F. Hohman, M. Kahng, J. Morgenstern and D. H. Chau, "FairVis: Visual analytics for discovering intersectional bias in machine learning," in *IEEE Conference on Visual Analytics Science and Technology*, 2019.

[114] Y. Chung, T. Kraska, N. Polyzotis, K. H. Tae and S. E. Whang, "Slice finder: Automated data slicing for model validation," in *IEEE 35th International Conference on Data Engineering*, 2019.

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

[116] A. B. Yoo, M. A. Jette and M. Grondona, "SLURM: Simple Linux Utility for Resource Management," in *9th International Workshop on Job Scheduling Strategies for Parallel Processing (JSSPP)*, 2003.

[117] "Cloudify," [Online]. Available: http://cloudify.co/.

[118] T. Kiss, P. Kacsuk, J. Kovacs, B. Rakoczi, A. Hajnal, A. Farkas, G. Gesmier and G. Terstyanszky, "MiCADO - Microservice-based Cloud Application-level Dynamic Orchestrator," *Future Generation Computer Systems,* vol. 94, pp. 937-946, 2019.

[119] INDIGO consortium, "The INDIGO-DataCloud project," [Online]. Available: https://www.indigo-datacloud.eu/ .

[120] D. Milojicic, "The edge-to-cloud continuum," *IEEE Computer,* vol. 53, no. 11, p. 16–25, 2020.

[121] L. Baresi, D. F. Mendonça, M. Garriga, S. Guinea and G. Quattrocchi, "A unified model for the mobile-edge-cloud continuum," *ACM Trans. Internet Technol,* vol. 19, no. 2, 2019.

[122] M. A. Tamiru, G. Pierre, J. Tordsson and E. Elmroth, ""“Instability in geo-distributed kubernetes federation: Causes and mitigation," in *28th Int. Symp. on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems*, 2020.

[123] L. Larsson, W. Tärneberg, C. Klein, E. Elmroth and M. Kihl, ""“Impact of etcd deployment on kubernetes, istio, and application performance," *Softw Pract Exp,* vol. 50, no. 10, p. 1986–2007, 2020.

[124] L. Osmani, T. Kauppinen, M. Komu and S. Tarkoma, "Multi-cloud connectivity for kubernetes in 5g networks," *IEEE Commun. Mag.,* vol. 59, no. 10, p. 42–47, 2021.

[125] L. Leong, "Comparing cloud workload placement strategies," Gartner Research, Tech. Rep., https://www.gartner.com/en/documents/3990249/comparingcloud-workload-placement-strategies, 2020.

[126] D2IQ, "Multi-cluster management: Reduce overhead and redundant efforts," D2IQ, Tech. Rep, https://d2iq.com/resources/cheat-sheet/multi-clustermanagement-reduce-overhead-and-redundant-efforts , 2021.

[127] P. Mell and T. Grance, "The NIST definition of cloud computing," NIST, Tech. Rep., https:// nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-145.pdf, 2011.

[128] A. Yousafzai, et al., "Cloud resource allocation schemes: Review, taxonomy, and opportunities," *Knowl. Inf. Syst,* vol. 50, no. 2, p. 347–381, 2017.

[129] R. Buyya, et al., "A manifesto for future generation cloud computing: Research directions for the next decade," *ACM Comput. Surv.,* vol. 51, no. 5, pp. 1-38, 2018.

[130] P. J. Maenhaut, B. Volckaert, V. Ongenae and F. D. Turck, "Resource management in a containerized cloud: Status and challenges," *J Netw Syst Manage,* vol. 28, no. 2, p. 197–246, 2020.

[131] S. M., "The emergence of edge computing," *IEEE Computer,* vol. 50, no. 1, pp. 30-39, 2017.

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

[133] S. Wang, J. Xu, N. Zhang and Y. Liu, "A survey on service migration in mobile edge computing," *IEEE Access,* vol. 6, 2018.

[134] P. Bosshart, D. Daly, G. Gibb, M. Izzard, N. McKeown, J. Rexford, C. Schlesinger, D. Talayco, A. Vahdat, G. Varghese and D. Walker, "P4: Programming Protocol-Independent Packet Processors," in *SIGCOMM Computer Communication Review*, 2014.

[135] C. Hopps, "Katran: A high performance layer 4 load balancer," 2019. [Online]. Available: https://github.com/facebookincubator/katran.. [Accessed 16 March 2023].

[136] Open Information Security Foundation, "Suricata," [Online]. Available: https://suricata.io/. [Accessed 16 March 2023].

[137] Sourcefire, "Snort," 2021. [Online]. Available: https://www.snort.org/. [Accessed 16 March 2023].

[138] The Linux Foundation, "Open vSwitch," 2018. [Online]. Available: https://www. openvswitch.org/. [Accessed 16 March 2023].

[139] A. Panda, S. Han, K. Jang, M. Walls, S. Ratnasamy and S. Shenker, "NetBricks: Taking the V out of NFV," in *12th USENIX Sympo- sium on Operating Systems Design and Implementation*, 2016.

[140] M. A. Vef, N. Moti, T. Süß, M. Tacke, T. Tocci, R. Nou, A. Miranda, T. Cortes and A. Brinkmann, "Gekkofs - a temporary burst buffer file system for HPC applications," *Journal of Computer Science and Technology,* vol. 35, no. 1, pp. 72-91, 2020.

[141] H. Luu, M. Winslett, W. Gropp, R. Ross, P. Carns, K. Harms, M. Prabhat, S. Byna and Y. Yao, "A multiplatform study of I/O behavior on petascale supercomputers," in *Proceedings of the 24th International Symposium on High-Performance Parallel and Distributed Computing, HPDC*, 2015.

[142] B. Xie, S. Oral, C. Zimmer, J. Y. Choi, D. Dillow, S. Klasky, J. Lofstead, N. Podhorszki and J. S. Chase, "Characterizing output bottlenecks of a production supercomputer: analysis and implications," in *ACM Transactions on Storage*, 2020.

[143] G. K. Lockwood, S. Snyder, T. Wang, S. Byna, P. Carns and N. J. Wright, "A year in the life of a parallel file system," in *International Conference for High Performance Computing, Networking, Storage and Analysis*, 2018.

[144] 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 Commun. Surveys & Tutorials,* vol. 23, no. 4, 2021.

[145] A. Kalia, M. Kaminsky and D. Andersen, "Design guidelines for high performance RDMA systems," in *USENIX Annual Technical Conference*, 2016.

[146] T. Odajima, Y. Kodama, M. Tsuji, M. Matsuda, Y. Maruyama and M. Sato, "Preliminary performance evaluation of the Fujitsu A64FX using HPC applications," in *IEEE international conference on cluster computing (cluster)*, 2020.

[147] C. Lattner and A. Vikram, "LLVM: A compilation framework for lifelong program analysis & transformation," in *International symposium on code generation and optimization*, 2004.

[148] Ferri, Pudil, Hatef and Kittler, "Comparative Study of Techniques for Large-Scale Feature Selection," in *Machine Intelligence and Pattern Recognition*, 1994.

[149] S. Zeuch, B. Del Monte, J. Karimov, C. Lutz, M. Renz, J. Traub, S. Breß, T. Rabl and V. Markl, "Analyzing efficient stream processing on modern hardware," in *Proc. VLDB Endow.*, 2019.

[150] M. Lapegna, V. Mele and D. Romano, "Clustering Algorithms for Enhanced Trustworthiness on High Performance Edge Computing Devices," *Electronics,* vol. 12, no. 7, 2023.

[151] J. Gedeon, F. Brandherm, R. Egert, T. Grube and M. Mühlhäuser, "What the Fog? Edge Computing Revisited: Promises, Applications and Future Challenges,," *IEEE Access,* vol. 7, pp. 152847-152878, 2019.

[152] C. Hong and B. Varghese, "Resource Management in Fog/Edge Computing: A Survey on Architectures, Infrastructure, and Algorithms," *ACM Comput. Surv.,* 2019.

[153] K. Fu, W. Zhang, Q. Chen, D. Zeng and M. Guo, "Adaptive Resource Efficient Microservice Deployment in Cloud-Edge Continuum," *IEEE Transactions on Parallel and Distributed Systems,* vol. 33, no. 8, pp. 1825-1840, 2022.

[154] G. Russo Russo, V. Cardellini and F. Lo Presti, "Hierarchical auto-scaling policies for data stream processing on heterogeneous resources," *ACM Transactions on Autonomous and Adaptive Systems,* 2023.

[155] P. Patros, J. Spillner, A. Papadopoulos, B. Varghese, O. Rana and S. Dustdar, "Toward sustainable serverless computing," *IEEE Internet Computing,* 2021.

[156] E. Sciacca, et al., "An integrated visualization environment for the virtual observatory: Current status and future directions.," *Astronomy and Computing ,* vol. 11, pp. 146-154, 2015.

[157] F. Vitello, et al., "Vialactea visual analytics tool for star formation studies of the galactic plane.," *Publications of the Astronomical Society of the Pacific,* 2018.