eFlows4HPC

Enabling dynamic and Intelligent workflows in the future EuroHPC ecosystem

D1.5 Interfaces and final software stack release

Version 1.0

Documentation Information

Contract Number	9555558
Project Website	www.eFlows4HPC.eu
Contractual Deadline	31.11.2023
Dissemination Level	PU
Nature	Other
Author	Jorge Ejarque (BSC)
Contributors	Jedrzej Rybicki (FZJ),Claudia Comito (FZJ), Yolanda Becerra (BSC), Anna Queralt (BSC), Alessandro D'Anca (CMCC), Sonia Scardigno(CMCC), Salvatore Giampà (DtoK), Jose Flich (UPV)
Reviewer	François Exertier (Atos)
Keywords	Requirements, Architecture, workflows

This project has received funding from the European High-Performance Computing Joint Undertaking (JU) under grant agreement No 955558. The JU receives support from the European Union's Horizon 2020 research and innovation programme and Spain, Germany, France, Italy, Poland, Switzerland, Norway.

Change Log

Version	Description Change
V0.1	Proposed Table of Contents
V0.2	Input from the different components Software Stack and Implementations
V0.3	Update of the Executive summary, introduction and Release infrastructure.
V0.4	Fixes for internal review
V1.0	Final version

Table of Contents

1 Executive Summary

This deliverable provides the final release of the eFlows4HPC software stack. This release consists of the source code implementation of the software stack components which are available in different public repositories in the eFlows4HPC Github organization (https://github.com/eflows4hpc/). Apart from the github repository, the latests releases of the components have been uploaded to the eFlows4HPC Zenodo Community (https://zenodo.org/communities/eflows4hpc/) in order to facilitate the proper reference and FAIR principle of the Software releases.

The main documentation of the software stack is provided through the Read the Docs documentation framework (https://eflows4hpc.readthedocs.io/). The modality adopted for this deliverable allows providing its contents online and evolving them as a live document as the project development proceeds. Releases to the document have been performed for each specific milestone which can be directly seen in the Read the Docs document. The version corresponding to this deliverable is version 3.0.

Apart from the source code and documentation, we have created deployment environments at the cloud environments of the Jülich and Barcelona Supercomputing Centers where we have deployed the Gateway services of the software stack. One environment is dedicated to development purposes and another for the validation from the use cases. Moreover, we have different supercomputers from JSC, CMCC and BSC as explained in section 3.4

This deliverable updates deliverable D1.4. In this sense, the introduction Section 2 is basically identical; Section 3 includes the Zenodo Community and the description of the HPC Clusters used for testing and validation, in addition to the Github repository to store the source code and ReadTheDocs for managing the Software Stack documentation; The main change in Section 4 i is focussed in the last release of the Software Stack, where we have included the Model Repository and integrated Unicore as alternative middleware to access the HPC systems; Finally, Section 5 describes the modifications performed in the software stack components during the last iteration of project.

2 Introduction

WP1 encompasses integration and development activities in order to provide programming interfaces to support workflows that integrate HPC, high-performance data analytics and machine learning approaches. In particular, Task 1.3 focuses on the identification and design of the necessary programming interfaces for the integration of HPC, data analytics and machine learning in a single workflow; Task 1.4 focuses on implementing the necessary interactions between the components for a smooth execution of the workflows, and deploying the required infrastructure to ensure a complete software stack integration; Task 1.5 focuses on the definition and implementation of the HPC Workflows-as-a-Service methodology; and finally, Task 1.6 focuses on the repositories to enable the reusability of workflows and their results.

This deliverable releases the final version of the results obtained in tasks 1.3, 1.4, 1.5 and 1.6 which composes the final release of the eFlows4HPC Software Stack.

3 Release infrastructure

The eFlows4HPC Software Stack is released as a set of tools, services, cloud resources and usage documentation to facilitate the open-source distribution, installation and deployment of the Software Stack components. The following paragraphs describe the different components of the release infrastructure.

3.1 Github Organization

To manage the source code of the implemented components, documentation and other code examples, we have set a Github organization available through the following link:

https://github.com/eflows4hpc/.

It provides a public space to manage different git repositories to store open-source code, package repositories to store binaries and container images, and the *github* teams working in the different implementations of the project. In this organization, we have stored the open-source releases of the components fully implemented in eFlows4HPC as well as forks of components which are modifications done in the eFlows4HPC project but not merged in the master branch of the original component. These repositories are pinned to appear at the top part of the main organization page.

In addition to component repositories, it also contains a repository to store the sources and build scripts for the on-line documentation of the eFlows4HPC Software Stack which is published in the *Read the Docs* platform.[1]

3.2 Zenodo Community

Zenodo is a general-purpose open repository that allows researchers to deposit research papers, data sets, research software, reports, and any other research related digital artifacts. When submitting an artifact in Zenodo, the submitter has to fill in some metadata to specify a description of the artifact, the authors, license, funding agencies, keywords and other metadata to fulfill the open access principles. After the submission, a persistent digital object identifier (DOI) is assigned to the artifact, which makes it easily citable.

To store the final releases of the software generated or modified by the project and make them citable and easily findable, we have created the eFlows4HPC Zenodo Community. This community can be found in the following link:

https://zenodo.org/communities/eflows4hpc/

3.3 Read the Docs documentation

Read the Docs is a platform to simplify software documentation by automating building, versioning, and free hosting of open source software. We have created a documentation project for eFlows4HPC in *Read the Docs*, with the goal of providing the users with the most recent version of the documentation in accordance with the modifications done during the implementation of the project. This eFlows4HPC documentation project is linked to the documentation repository stored in the eFlows4HPC github organization. This repository can be found in the following link:

https://github.com/eflows4hpc/documentation

[1] https://readthedocs.org/

For the different version branches created in the repository, a new version is automatically created in the documentation. The master branch of the repository refers to the latest version of the documentation. Everytime a "push" or "merge" is performed in one of these branches, the corresponding version is automatically updated. The version available for this deliverable is version 3.0.

The generated on-line documentation and the different versions can be found in the following link:

https://eflows4hpc.readthedocs.io/

The documentation git repository contains the *doc* section where the content material is stored and the *Readme* file together with the setup files. All the document sections are in the *doc/sources/Sections* folder which stores the source files of the documentation following the reStructuredText format.[2]

The repository and its generated on-line documentation are structured in five sections. The *eFlows4HPC Overview* section provides an overview of the project. The *Software Stack* section provides an overview of the architecture and the list of the eFlows4HPC components with a short description together with useful information on how to find the source code, the component installation and usage guidelines. Afterwards, the *Programming Interfaces for integrating HPC and DA/ML workflows* introduces the workflow programming interfaces available to integrate different software in a single workflow. Then, the *HPCWaaS Methodology* introduces the HPC Workflows as a Service methodology proposed by the eFlows4HPC project, which allows developers to easily deploy workflows in HPC clusters and facilitate the executions of the deployed workflows. Finally, the *Step by step example* section provides an example on how to use the different components together to develop and execute a workflow using the eFlows4HPC components. In this example, the Pillar I workflow is used as reference.

3.4 Deployment Environments

To facilitate the component integration and update of the Software Stack, we are using two cloud environments: the *HDF Cloud*[3] at Jülich Supercomputing Center and *nCloud*[4] at Barcelona Supercomputing Center. Both Cloud services are managed by OpenStack. Partners have created Virtual Machines and deployed gateway services of the eFlows4HPC Software Stack. The *HDF Cloud* is used for the development, testing and integration of the eFlows4HPC Software Stack, while the *nCloud* is used by Pillar's workflow developers to validate the implementations in a stable environment. The end-points of the deployed services can be found in the Software Stack page of the documentation.

Apart from these Cloud infrastructures to deploy the Gateway services, we have also used an heterogeneous set of HPC systems where we have tested the implemented workflows. From BSC, we have used the MareNostrum 4[5] where each node contains 48 Intel Skylake cores with local SSD disk and OmniPath interconnect; Nord3v2[6] where each node contains 16 Intel Sandybridge cores

[2] https://docutils.sourceforge.io/rst.html

[3] https://hdf-cloud.fz-juelich.de

[4] https://ncloud.bsc.es

[5] https://www.bsc.es/supportkc/docs/MareNostrum4/overview

[6] https://www.bsc.es/supportkc/docs/Nord3v2/overview

with local HHD storage and Infiniband interconnect; CTE-AMD[7] where each node contains 64 AMD EPYC (zen2) cores and 2 AMD Radeon Instinct GPUs with local SSD storage and Infiniband interconnect; and CTE-Power[8] where each node contains 40 IBM Power9 cores and 4 NVIDIA V100 GPUs with local NVMe and SSD storage. All the BSC machines are accessed through SSH/SCP protocols, have the GPFS as a shared file system and compute nodes are managed by Slurm.

From CMCC, we have used Zeus[9] where each node contains 18 Intel Skylake cores with Infiniband interconnect and GPFS as a shared file system. Zeus is accessed through SSH/SCP protocols and compute nodes are managed by LSF.

From JSC, we have used JUSUF[10] where each node contains 64 AMD EPYC (zen2) cores and a NVIDIA V100 GPU with local NVMe and SSD storage, Infiniband interconnect and GPFS as a shared file system. JUSUF can be accessed through UNICORE and computing nodes are managed by Slurm.

4 Implementations

This section introduces the material provided in the Read the Docs repository about the implementations included in the eFlows4HPC software stack release. The project is delivering the eFlows4HPC software stack that integrates different components to provide an overall workflow management system. One of the core functionalities of the software stack is the definition of complex workflows that combine HPC, HPDA and ML frameworks and the integration of large volumes of data from different sources and locations. Using this software stack, the project builds an HPC Workflow as a Service (HPCWaaS) platform to facilitate the reusability of these complex workflows in federated HPC infrastructure. The sections of the Read the Docs repository that describe the adopted solutions can be found at the already mentioned link: https://eflows4hpc.readthedocs.io/. Apart from the introductory one (eFlows4HPC), they are as follows.

4.1 Software Stack

This section presents an overview of the eFlows4HPC software stack release and its deployment. The software components of the designed stack are listed and introduced. In particular the software stack components are organized in three groups according to the deployment model. First, the Gateway services are components deployed outside the HPC infrastructure which are in charge of managing the workflow lifecycle, allowing users to store the workflow descriptions and performing the deployment and execution of their workflows in the selected computing infrastructure. Second, the Runtime components are deployed inside the computing infrastructure and are in charge of orchestrating the computations and data management during the workflows execution. Finally, DA/ML frameworks are components deployed with the workflows which are in charge of performing the Data Analytics and ML parts of a workflow. In this last iteration, we have

[7] https://www.bsc.es/supportkc/docs/CTE-AMD/overview

[8] https://www.bsc.es/supportkc/docs/CTE-POWER/overview

[9] https://www.cmcc.it/super-computing-center-scc

[10] https://apps.fz-juelich.de/jsc/hps/jusuf/cluster/configuration.html

released the two components that were missing in the architecture: the Model Repository and the integration of Unicore.

(See: https://eflows4hpc.readthedocs.io/en/latest/Sections/01_Software_Stack.html).

4.2 Programming Interfaces for integrating HPC and DA/ML workflows

This section introduces the programming interfaces and explains how the eFlows4HPC programming interface has been designed to reduce the effort required to integrate different frameworks in a single workflow. The integration is divided in two parts: Software Invocation Management and Data Integration. In the first iteration, we have defined the software invocation descriptions and we have extended the PyCOMPSs programming model and runtime accordingly. In the second iteration we have implemented the data transformations, and similarly we have extended the PyCOMPSs programming model and runtime to support them. In the final iteration, we have extended the data transformations to support new data types such as directories.

(See:https://eflows4hpc.readthedocs.io/en/latest/Sections/02_Programming_Interfaces.html)

4.3 HPCWaaS methodology

In this section, the HPC Workflow as a Service (HPCWaaS) methodology that applies the usage of the Functions as a Service (FaaS) model in Cloud environments for workflows in HPC systems is sketched. This model defines different main roles. In particular, the function developer is in charge of developing and registering the function in the FaaS platform, and the final user executes the deployed function using a REST API. In the case of workflows running in HPC systems, we can find similar roles. In fact, the workflow developer is in charge of developing and deploying the workflow in the computing infrastructure, and the users' communities execute the workflow and collect its results to advance in their scientific goals. This section in the Read the Docs repository includes two subsections describing the Development interface provided by Alien4Cloud and the Execution API.

(See:https://eflows4hpc.readthedocs.io/en/latest/Sections/03_HPCWaaS_Methodology.html)

4.4 Step-by-Step Example

This section was introduced during the second iteration and it has been updated in the last iteration. It describes a usage example of a real scientific workflow that can be implemented, deployed and executed using the eFlows4HPC Software Stack and the HPC Workflows-as-a-service methodology. To illustrate it, we have used the Pillar I workflow which combines different executions to create a Reduced Order Model from a Full Order Model. This workflow combines the Full Order Model simulations, performed by the Kratos Multiphysics software, with the ML algorithms implemented by the dislib framework which are used to compute the Reduced Order Model. The computational workflow has been implemented with PyCOMPSs. The creation of the container Images for this workflow has been performed by the Container Image Creation Service. Finally, the Data Logistic service is used to perform the data pipelines required for deploying and executing the workflow. At the deployment phase, it transfers the container image to the HPC site, while at the execution phase, it performs the deployment of the configuration files and meshes required for the FOM simulations from a B2DROP repository to the HPC cluster and uploads of the obtained Reduced Order Model to the Model Repository.

The overall workflow lifecycle is managed with the HPCWaaS interfaces. The workflow is described as a TOSCA topology including the relationship between the required HPC sites, container creations, data pipelines and computations using the Alien4Cloud component. This topology allows the orchestration, deployment and execution of the workflow. Once it is described it can be stored in the Workflow Registry and deployed using Yorc.

After a successful deployment, the service is ready to be used by the final users. Users can perform the workflow execution using the Execution API.

All the details of the steps required to implement, deploy and execute the workflow are explained in the Read the Docs repository. A video has also been added during the last iteration.

(See: https://eflows4hpc.readthedocs.io/en/latest/Sections/04_Usage_Example.html)

5 Software stack components updates

This section provides details about the implementations or modifications performed in the different components during the eFlows4HPC project. In the previous deliverables, we presented the components introduced in the first and second iteration and the changes performed to the existing components. In this deliverable, we provide the details of the modifications performed in the components during the last iteration of the project. The Ystia Orchestrator and Data Catalogue components have not been modified, so this is the reason why they are not appearing in the following paragraphs.

5.1 Gateway Services

5.1.1 Workflow Registry

The Workflow Registry is a new component developed in the second reporting period of the eFlows4HPC project. This component is developed on top of a github repository storing the descriptions of the workflows implemented in the project. More details about this component are given in the deliverable D1.3[11]. In the last period of the project, the Workflow Registry has been populated with the latest versions of the workflows of the different pillars. We have also included example workflows with specific tools and workflows implemented in collaboration with other CoEs and institutions.

5.1.2 Software Catalog

The Software Catalog is a new component developed in the second reporting period of the eFlows4HPC project. This component is developed on top of a github repository which is storing the descriptions of the possible software invocations as json files and the installation descriptions as spack packages. More details about this component are given in the deliverable D1.3. In the last period of the project, the Software Catalog has been populated with the descriptions of the eFlows4HPC DA/ML tools and other software used in the implemented workflows.

[11] eFlows4HPC consortium, “Deliverable D1.3 Revision of Requirements and Architecture Design” August 2022

5.1.3 Alien4Cloud

The Alien4Cloud software component itself has not been modified in the context of eFlows4HPC. Just some bugs found in the definition of the use cases have been fixed. Instead we developed a set of TOSCA components to integrate eFlows4HPC services within TOSCA workflows.

In the last period of the project, the DLS TOSCA components have been extended to support the new data pipelines implemented in the DLS for supporting new storage protocols as well as storing and serving the models in the Model Repository.

5.1.4 Workflow Execution Service

The workflow execution service is developed from scratch specifically for the eFlows4HPC project.

It is mainly composed of a REST API and its CLI companion.

The development effort was mainly on:

- designing and implementing the REST API
- implementing an interface with Alien4Cloud to manage workflows
- implementing the CLI
- implementing an interface to interact with Hashicorp Vault to manage secrets

The development of this service started at the end of the first reporting period of the project and continued during the second and third reporting period. In this last period, we have extended the component with the integration with the Unity Single-Sign-On provider.

5.1.5 Container Image Creation Service

The Container Image Creation Service is a new component developed in the second reporting period of the eFlows4HPC project. This component leverages HPC Builders and Multi-platform container builders in order to create images tailored for specific HPC clusters. This component has been implemented as a Flask service which offers a REST API for requesting the image creation, checking the creation status and downloading the created image.

During the last period of the project, we have created a web graphical user interface, which allows the user to monitor the image creation requests, the created images and API credentials. We have also extended the image creation environment to support apt and Python pip packages. This will speed up the installation of packages that do not require HPC features. Finally, we have also implemented a standalone version in the form of a library which can be used without deploying this component as a service. It can be useful for CI/CD purposes.

5.1.6 Data Logistics Service

The Data Logistics Service (DLS) is one of the central gateway services in the eFlows4HPC stack. It is responsible for data movements between community repositories and computing facilities (HPC/HTC/Clouds). The data movements are formalized as open-source *pipelines* written in an easy-to-use Python framework. This allows the users to share their own ways of moving and accessing the data, modify the existing ones to accommodate their special needs, and improve the reproducibility of the research by capturing the important but often overlooked step of the data movements.

The Data Logistics Service is based on open source Apache Airflow (an industry de-facto standard) for data movements. This service was created for the eFlows4HPC project and the main developments have been on the integration of the service with the other parts of the software

stack. Furthermore, a set of data movement pipelines for the project users was created. There were also changes in the user interface and an ongoing effort to enable a Single-Sign-On to the service. For the third release, new data movement pipelines have been added, the service was upgraded to the newest Apache Airflow version, and we have integrated it with the project SSO solution.

5.1.7 Model Repository

The service provides a means for scientific users to store serialized models along with the metadata describing their machine learning/AI experiments. The model repository service is a new addition to the eFlows4HPC software stack in the 2nd year of the project. The model repository is based on the open source MLFlow software, which is widely used in the AI/ML community.

We envision the following scenarios for the model repository. In case of interactive access (e.g. via Jupyter notebook), model training parameters, evaluation metrics and other metadata can be uploaded directly into the repository. Alternatively, if model training is done offline, locally collected metadata is uploaded to the model repository during the stage-out phase using the Data Logistics Service. Finally, users in the project and beyond can browse the model repository, compare the metadata, and download the models for reuse. The current pilot version of the Model Repository supports these three use cases and will be evaluated by users in the coming months to address additional requirements. In addition, we have prepared examples for all use cases and integrated the model repository into the eFlows4HPC software stack by implementing the required DLS pipeline and TOSCA component.

5.1.8 Unicore

In this final phase of the project, we have included Unicore as another possibility to access the HPC resources in addition to the existing support for the SSH/SCP protocols. Unicore provides seamless, unified access to HPC resources. This unified access includes the data management and execution interfaces. To integrate Unicore into the eFlows4HPC software stack, we have modified the eFlows4HPC data management and the execution mechanism to support the interaction with the Unicore services.

On the one hand, we have implemented DLS pipelines to stage-in data through the Unicore interface. Currently, it is possible to stage-in data from a WebDAV interface (such as the one provided by B2DROP), which is the most popular interface used in the eFlows4HPC workflows. It should be noted that most of the sites offering Unicore-based access also accept alternative protocols for stage-in. This is the case for the FZJ infrastructure, where eFlows4HPC users can use SSH or ObjectStore interfaces (already covered by the DLS) depending on their needs.

On the other hand, we have modified the PyCOMPSs CLI to support the execution of workflows using the Unicore interface in addition to the existing support for SSH. The PyCOMPSs CLI is used by Yorc to execute the workflows from the HPC WaaS interface.

5.2 Runtime Components

5.2.1 PyCOMPSs

COMPSs is a task-based programming model which provides parallel execution of applications on distributed systems. Its model abstracts the application from the underlying distributed infrastructure, allowing it to be portable between infrastructures with diverse characteristics. PyCOMPSs is the Python binding of COMPSs.

When developing with PyCOMPSs, distribution of the data, task scheduling, data dependency between tasks, and fault tolerance issues are hidden to the user and are the responsibilities of the COMPSs Runtime. The COMPSs Runtime is also able to react to task failures and exceptions in order to adapt the behavior accordingly.

In the last period of the project, PyCOMPSs have been extended in several ways to support the functionalities required in the Pillar workflows and the eFlows4HPC deployment model:

- The COMPSs runtime has been extended to support multi-node executions (MPI, MPMD,...) when using container deployments. In container deployments, normal task processes are executed inside the container. However, in multi-node task executions, processes (such as MPI processes) must be started in other nodes and it must be performed using containers.
- The PyCOMPSs CLI has been extended to support the executions of PyCOMPSs workflows using Unicore.
- The @dt decorator has been extended to support conversion to directory data types. It is required for the new Pillar I ROM which are serialized in different files inside a folder.
- The PyCOMPSs serializer has been extended to support EDDL objects and Pytorch tensors.
- A Software cache for objects in the GPU has been included in the worker part of the PyCOMPSs runtime to improve the re-usage of data in GPU tasks.

These extensions have been included in the releases 3.2 and 3.3 of COMPSs.

5.2.2 dataClay

dataClay is a distributed object store with active capabilities. It is designed to hide distribution details while taking advantage of the underlying infrastructure, be it an HPC cluster or a highly distributed environment such as edge-to-cloud. Objects in dataClay are enriched with semantics, giving them a structure as well as the possibility to attach arbitrary user code to them. In this way, dataClay enables applications to store and access objects in the same format they have in memory (Python or Java objects), also allowing them to execute object methods within the store to exploit data locality. This active capability minimizes data transfers, as only the results of the computation are transferred to the application, instead of the whole object.

dataClay implements the *Storage Runtime Interface* that PyCOMPSs can use to enhance data locality of parallel and distributed applications. This implementation hints the runtime scheduler to assign tasks on the same nodes where dataClay stores the needed data, and allows to avoid the cost of serializing this data when it is accessed from several tasks.

During the previous project iterations, dataClay was extended with the *SpIIter*[12] functionality, which enables efficient iteration of distributed datasets for their processing. *SpIIter* provides data partitions that reduce the number of tasks and, as a consequence, also scheduling and invocation overheads are reduced.

During the third iteration, the implementation of *SpIIter* takes advantage of optimized metadata management, reducing overheads in *SpIIter* itself. In a similar way, the functionalities in the *Storage Runtime Interface* used by PyCOMPSs during the scheduling of tasks (such as creation of

[12] A. Barceló, A. Queralt, T. Cortes: Enhancing iteration performance on distributed task-based workflows. Future Gener. Comput. Syst. 149: 359-375 (2023)

read-only replicas or read-write versions of objects) have been re-implemented leveraging metadata optimizations, reducing overheads in their execution.

Additionally, the dataClay classes definition now supports extra hints, given by the developer, to control the data placement of attributes. This is achieved through standard typing information (available in Python 3) and context-specific metadata (a feature available since Python 3.9). Currently two types of hints for attributes are supported in dataClay:

- LocalOnly: avoids serialization of a parameter across nodes and persistent storage, which can be leveraged to implement volatile attributes that don't need to be persistent, or node-local caches.

- InNVM: transparently places numpy arrays in NVRAM (persistent memory devices) without developer intervention. The mechanism used for the placement is the one we evaluated in the context of WP3, and now these annotations export the functionality to the application developer in a convenient way, improving programmability and portability.

These extensions have been included in the releases 3.1 of dataClay.

5.2.3 Hecuba

Hecuba is a set of tools and interfaces that implement a simple and efficient access to data stores for big data applications. One of the goals of Hecuba is to provide programmers with an easy and portable interface to access data. This interface is independent of the type of system and storage used to keep data, enhancing the portability of the applications. Using Hecuba, the applications can access data like regular objects stored in memory and Hecuba translates the code at runtime into the proper code, according to the backing storage used in each scenario.

Hecuba also implements the *Storage Runtime Interface* that PyCOMPSs can use to enhance data locality of parallel and distributed applications. This implementation hints the runtime scheduler to assign tasks on the nodes where Hecuba stores the needed data, and allows to avoid the cost of serializing this data.

At the beginning of the project, Hecuba offered the user an interface for Python programs. It also implemented a proof of concept that allowed its use from C++ programs. During the first reporting period of the project, a C++ interface has been defined to facilitate the use of Hecuba from the use cases of the project. And during the second reporting period of the project, we designed and implemented into Hecuba a mechanism that provides synchronization between producer and consumers to support online data analysis, which was a new requirement identified in the Pillar II use case.

During the third period, we have followed three main working lines on Hecuba. The first one has focused on improving the C++ interface to provide a better experience to the programmers. This new interface minimizes the number of Hecuba-specific code lines in C++ applications, speeding up the productivity and learning curve of the programmers. The second one consists of extending the automatic deployment of Hecuba to support container-based environments, which was a requirement to enable the execution of the ESM Dynamic Workflow of Pillar II in a container-based environment. The third one consists of analyzing and improving the performance of the C++ interface. To this extent, we have added to Hecuba the instrumentation code to generate Extrae execution traces. The analysis of these traces leads us to improve some internals of Hecuba as, for example, the management of the concurrency and the internal data model to store *StorageNumpys*.

5.3 ML and DA Frameworks

5.3.1 dislib

The Distributed Computing Library (dislib) is a library that provides various distributed machine-learning algorithms. It has been implemented on top of PyCOMPSs, with the goal of facilitating the execution of big data analytics algorithms in distributed platforms, such as clusters, clouds, and supercomputers. Dislib comes with two primary programming interfaces: an API to manage data in a distributed way and an estimator-based interface to work with different machine learning models. For the first interface, Dislib provides the distributed array (ds-array) that enables the distribution of the data sets in multiple nodes of a computing infrastructure.

In the last period of the project, we have extended the algorithms of the dislib including an SVD approximation using the Lanczos method and another one using a randomized approach, and a more scalable implementation of the Random Forest Regressor, required for Pillar I and III respectively. In addition to these extensions, we have improved the model saving feature with a compression mechanism to reduce the size of the models.

These new algorithms are included in the release 0.9.0 of dislib.

5.3.2 EDDL

EDDL is the European Distributed Deep Learning library. It is mainly written in C++ and targets CPU, GPU and FPGA devices. The main goal of EDDL is to serve as a European Library for training and inference operations over Deep Learning neural networks.

During the third reporting period of the project, EDDL has been improved with the support of specific datasets from the project. The main limitation faced by EDDL is the lack of standard support of current formats used in ML frameworks. In addition, the library has been tested during this period with multi-GPU systems.

5.3.3 Heat

Heat is a flexible and seamless open-source software for high performance data analytics and machine learning. It provides highly optimized algorithms and data structures for tensor computations using CPUs, GPUs and distributed cluster systems on top of MPI. Heat is used in a variety of contexts, from atmospheric modeling to neuroscience, whenever the capabilities of a Python-based analysis pipeline are severely impaired by single-node RAM constraints, i.e. when either the numerical (array-like) data, or required operations on the data, require more RAM than is available on a single node (CPU/GPU) of a cluster. Heat implements PyTorch- and MPI-based data decomposition and necessary communication under the hood, while the front end strives to implement the Python array API, and is therefore fully interoperable with the existing Python data science ecosystem.

During the third period, Heat v1.3 was released, including the final implementation of memory-distributed truncated SVD. This dimensionality-reduction method at the basis of PCA (Principal Component Analysis) is now available to users as “Hierarchical SVD” in the *linalg* module (*heat.linalg.hsvd*), and enables hardware-accelerated decomposition of massive low-rank matrices requiring multi-node, multi-GPU memory resources.

During this period, we have also implemented support for the ROCm environment and usage of AMD GPUs. Multi-node, multi-GPU support of ROCm is still experimental as of this writing.

Spack packages and containerized versions of Heat are now available on our repository[13] to facilitate usage within HPC workflows.

A list of all new features is available on the Heat repository under CHANGELOG.

5.3.4 Ophidia

The Ophidia HPDA framework, developed by CMCC, addresses challenges related to management and analysis of scientific multi-dimensional data by joining HPC paradigms and Big Data approaches. The framework is primarily used in the context of climate sciences, although it has also been successfully exploited in other domains (e.g., astronomy, seismology, and smart cities). More specifically, Ophidia provides in-memory, parallel, server-side data analysis and I/O and an internal storage model, based on the datacube abstraction inherited from the On-line Analytical Processing (OLAP) systems, and a hierarchical organization to partition and distribute large amounts of multi-dimensional scientific data. The platform also provides the Python bindings called PyOphidia to easily integrate Ophidia functionalities into Python applications. Thanks to its flexibility, in the eFlows4HPC project, Ophidia is used for the processing stages of Pillar II related to climate extreme events and Pillar III related to Tsunamis.

To better support the two Pillar use cases several extensions and optimizations have been carried out throughout the project. Besides some bug fixing and performance improvements, the key developments for the last release in eFlows4HPC include:

- Extension of a parallel operator for loading data from multiple NetCDF files using a text file for the list of input paths.
- Bug fixing of operators implemented for supporting Pillar use cases (e.g. *oph_mergecubes2* and *oph_importncs* operators, *oph_interlace2* primitive).
- New Ophidia scripts[14] to simplify the configuration of the whole framework on HPC systems. Moreover, the scripts for the integration with PyCOMPSs have been improved to better handle multi-node deployment[15].
- Improved PyOphidia[16] documentation to better support the integration by users from different pillars, as well as potential external users.

The full set of changes has been included in the releases 3.0 on the related eFlows4HPC GitHub repositories.

5.3.5 ParSoDA

ParSoDA (Parallel Social Data Analytics) is a high-level programming library aiming at simplifying the development of parallel and distributed social media mining applications executed on High

---

[13] https://github.com/helmholtz-analytics/heat/tree/release/1.3.x/docker

[14] https://github.com/eflows4hpc/ophidia-server/tree/v3.0/etc/config

[15] https://github.com/eflows4hpc/ophidia-server/tree/v3.0/etc/script/multinode

[16] Donatello Elia, Cosimo Palazzo, Sandro Fiore, Alessandro D'Anca, Andrea Mariello, Giovanni Aloisio,

PyOphidia: A Python library for High Performance Data Analytics at scale, SoftwareX, Volume 24,

2023, https://doi.org/10.1016/j.softx.2023.101538.

Performance Computing systems. ParSoDA defines a general framework for designing and executing social media analysis applications as a sequence of defined steps (data acquisition, filtering, mapping, partitioning, reduction, analysis, and visualization), which can make use of a predefined (but extensible) set of functions. Thus, an application developed with ParSoDA is expressed by a concise code that specifies the functions invoked at each step. In this way, data scientists and analysts having limited programming skills, especially with regard to parallel programming, can efficiently design and execute large-scale data analysis applications dealing with big data.

Originally, ParSoDA was developed in Java and was capable of supporting the execution of applications on top of both Apache Hadoop and Apache Spark. In the eFlows4HPC project, we have rewritten ParSoDA using Python to support the COMPSs runtime through its binding PyCOMPSs for addressing the functionalities required by the Pillar Workflows.

In the third reporting period of the project, we conducted a series of performance evaluations. The tests have been conducted on the Trajectory Mining application, by comparing PyCOMPSs and PySpark as runtime systems. The dataset size has been varied from 1 GB to 140 GB and the number of processors available for the runtimes varied from 8 up to 256 cores. For running the tests, a Docker Swarm cluster was used, which was composed of 1 master node and 4 worker nodes, of 64 processing units each.

The measured quantities are the following:

- **Response time:** for each step of the ParSoDA workflow we measured the corresponding execution time and, by adding them, the variation of the total execution time by varying the number of available processors.
- **Speed-up:** by defining a baseline for each runtime, we evaluated the speed increment by incrementing the number of processors.
- **Scalability:** we evaluated the increment of speed-up by incrementing the dataset size.

In order to be worthy of the complexity due to the potential availability of many runtime systems for ParSoDA-Py, setting up some integration tests for controlling correctness of the software components is a must. For this reason, we have defined some tests for checking correctness of newly added drivers or changes to the existing ones, by comparing their results with those from the *ParsodaSingleCoreDriver*.

Pre-built applications have been defined, in order to be exposed as ready-to-use functions available in the *parsoda.apps* package. Each application is defined as a function which accepts the domain and runtime parameters of the application and returns the corresponding SocialDataApp instance, ready to be executed. Finally, some improvements to the sentiment analysis steps have been done. In particular we specialized the filter used for selecting the topic of interest from the dataset. All these improvements are available in the online GitHub repository of the library at https://github.com/eflows4hpc/ParSoDA-Py.

6 Acronyms and Abbreviations

- AI - Artificial Intelligence
- API - Application Programming Interface
- CIC - Container Image Creation
- CLI - Command Line Interface
- CPU - Central Processing Unit
- D – deliverable
- DA - Data Analytics
- DAG - Directed Acyclic Graph
- DC - Data Catalogue
- DL - Deep Learning
- DLS - Data Logistics Service
- DMCF - Data Mining Cloud Framework
- EDDL - European Distributed Deep Learning library
- ETL - extract, transform, load
- FaaS - Function as a Service
- FAIR - Findable Accessible Interoperable Reusable
- FPGA - Field Programmable Gate Array
- GPU - Graphics Processing Unit
- GUI - Graphical User Interface
- Heat - Helmholtz Analytics Toolkit
- HPC – High Performance Computing
- HPCWaaS - HPC Workflow as a Service
- HPDA - High-performance Data Analytics
- IaaS - Infrastructure as a Service
- ID- Identifier
- JSON - JavaScript Object Notation
- KPI – Key Performance Indicator
- M - Month
- ML - Machine Learning
- MPI - Message Passing Interface
- MR - Model Repository
- NN - Neural Network
- NVRAM - Non-Volatile Random Access Memory
- OLAP - On-line Analytical Processing
- ParSoDA - Parallel Social Data Analytics
- PCA - Principal Component Analysis
- POSIX - Portable Operating System Interface
- PRACE - Partnership for Advanced Computing in Europe
- REST - Representational State Transfer
- SC - Software Catalog
- SCP - Secure Copy
- SSD - Solid State Disk
- SSH - Secure Shell
- SSO - Single Sign-On
- SVD - Singular Vector Decomposition
- TOSCA - Topology and Orchestration Specification for Cloud Applications

- UI - User Interface
- VM - Virtual Machine
- VPN - Virtual Private Network
- WP – Work Package
- WR - Workflow Registry