D1.4 Interfaces and iteration 2 software stack release

Version 1.0

Documentation Information

Contract Number	9555558
Project Website	www.eFlows4HPC.eu
Contractual Deadline	28.02.2023
Dissemination Level	PU
Nature	R
Author	Loïc Albertin (BULL-Atos)
Contributors	Rosa M. Badia (BSC), Jorge Ejarque (BSC), Yolanda Becerra (BSC), Anna Queralt (BSC), Alex Barceló (BSC), Domenico Talia (DtoK Lab), Jedrzej Rybicki (FZJ), Donatello Elia (CMCC), José Flich (UPV)
Reviewer	Sonia Scardigno (CMCC)
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	Preliminary version
V0.3	Revised version after review
V1.0	Final version formatted for submission

Table of Contents

1 Executive Summary

This deliverable provides the second version of the eFlows4HPC software stack. According to their deployment model, the software stack components have been in three types: Gateway services which are deployed outside the computing infrastructure to manage the workflow lifecycle; Runtimes which are deployed in the computing infrastructure to orchestrate the workflow computation and data management at execution time; and ML/DA frameworks which are deployed with the workflow to manage ML/DA operations. The source code implementation of the software stack components are available in the public Github project repositories (https://github.com/eflows4hpc/), and Section 5 of the document provide the details about what are the implementations/ modification introduced in eFlows4HPC project.

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 will be performed for each specific milestone which can be directly seen in the Read the Docs document. The version corresponding to this deliverable is version 2.0.

Apart from the source code and documentation, we have created deployment environments 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.

This deliverable updates deliverable 1.2. In this sense, sections 2 and 3 are basically identical. Section 4 includes updates and a new subsection 4.4. Section 5 is new and, for the different components, specifies whether they are new in the project. In case the component already existed before the project, the section describes how it has been updated during the 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 second version of the results obtained in tasks 1.3, 1.4, 1.5, and some partial results from task 1.6 which composes the Iteration 2 software stack release.

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 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

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 1.0.

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* 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*

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

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.

3.3 Deployment Environments

To facilitate the component integration and update of the Software Stack, we are using two cloud environments: the *HDF Cloud* (https://hdf-cloud.fz-juelich.de) at Jülich Supercomputing Center and *nCloud* (https://ncloud.bsc.es) 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.

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 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 second iteration, we have released some new components (Container Image Creation, Workflow Registry and Software Catalog) and several updates have been performed in the other components.

(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 this 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.

(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 is sketched that applies the usage of the Functions as a Service (FaaS) model in Cloud environments for workflows in HPC systems. 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 new section introduces 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. This workflow has been implemented with PyCOMPSs, the creation of the container Images for this workflow has been performed by the Container Image Creation Service and data pipelines for the container image deployment, the acquisition of the input mesh to perform the simulation on the HPC system and the upload of the obtained Reduced Order Model are managed by the Data Logistic Service.

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.

(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. For those components implemented from scratch, we have described the main available features.

5.1 Gateway Services

5.1.1 Data Catalogue

The Data Catalogue is a repository service that keeps track of information about data sets created and used in the project. It is primarily used in the Data Logistics Service to make the data movement pipelines generic. The pipelines accept records from Data Catalogue as input from which they retrieve information about the access methods (e.g. URLs) to facilitate the subsequent data movements. The second goal of the Data Catalogue was to increase the visibility of the data sets created in the project (e.g. through the execution of the workflows) and to facilitate the reuse. To this end, it is possible to use DLS pipelines to register the created data sets with the Data Catalogue in the stage-out phase.

The Data Catalogue is a new component developed in eFlows4HPC. It provides an API and a web interface, is able to store basic metadata about the respective data set, and supports easy searching through the records. The most notable feature implemented in the second reporting period is the aforementioned search and filter functionalities requested by the users.

5.1.2 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 which is storing the descriptions of the workflows implemented in the project. More details about this component are given in the deliverable D1.3.

5.1.3 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.

5.1.4 Alien4Cloud

The Alien4Cloud software component itself has not been modified in the context of eFlows4HPC.

Instead we developed a set of TOSCA components to integrate eFlows4HPC services within TOSCA workflows which are described below.

5.1.4.1 Data Logistics Service TOSCA component

The Data Logistics Service (DLS) TOSCA component defines a set of TOSCA node types that allows to trigger and monitor DLS pipelines from within TOSCA workflows.

Those components interact with the DLS REST API to trigger, monitor and manage the execution of data pipelines.

This set of components is composed of a hierarchy or TOSCA node types with specialized nodes like data stage-in, data stage-out or container image transfer inheriting from a generic node type allowing to trigger arbitrary data pipelines.

Most of these components were originally developed during the first reporting period of the project and updated during the second reporting period.

5.1.4.2 PyCOMPSs TOSCA component

The PyCOMPSs TOSCA component defines a TOSCA node type that allows to trigger and monitor PyCOMPSs jobs from within TOSCA workflows.

This node type interacts with PyCOMPSs and Slurm over SSH.

This TOSCA component was developed during the first phase of the project, but due to some limitations inherent to plain TOSCA implementations, it has been replaced in a second phase by a PyCOMPSs plugin for Yorc described in PyCOMPSs Ystia Orchestrator plugin.

This component was originally developed during the first reporting period of the project and updated during the second reporting period.

5.1.4.3 Cluster Environment TOSCA component

The Cluster Environment TOSCA component defines a TOSCA node type that allows defining cluster-specific properties like the address of the cluster login node or the PyCOMPSs version to use on this cluster within a TOSCA application.

The DLS TOSCA component and the PyCOMPSs Yorc plugin can use this component to retrieve cluster-specific information.

This component was developed during the second reporting period of the project.

5.1.5 Ystia Orchestrator

5.1.5.1 Improvements to the Ystia Orchestrator

While the Ystia Orchestrator existed before the eFlows4HPC project, by developing eFlows4HPC specific TOSCA components we identified and fixed a few bugs and limitations within the Ystia Orchestrator (Yorc).

Limitations were mainly focused on the TOSCA function expressiveness and the interaction with the Hashicorp Vault component used in this project to store secrets.

Most of these changes to Yorc were made during the second reporting period of the project when more complex use cases emerged.

5.1.5.2 PyCOMPSs Ystia Orchestrator plugin

The PyCOMPSs Ystia Orchestrator (Yorc) plugin is an evolution of the PyCOMPSs TOSCA component with a totally different implementation developed in the eFlows4HPC project. Instead

of being just a TOSCA description interpreted by Yorc in a generic way, it has a specific implementation within the orchestrator. This allows it to leverage all the capabilities offered by Yorc. This is particularly true when interacting with TOSCA workflows context variables like workflow inputs.

During this refactoring we also switched from an interaction with PyCOMPSs and Slurm over SSH to an interaction with a local PyCOMPSs CLI. By doing this, we removed a dependency to Slurm that allows now to target any cluster scheduler supported by PyCOMPSs.

The development of this plugin started at the end of the second reporting period of the project.

5.1.6 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 reporting period.

5.1.7 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.

5.1.8 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. Since the first release, new data movement pipelines have been added, some performance issues have been fixed, and a better integration with the Data Catalogue has been implemented.

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 behaviour accordingly.

In the eFlows4HPC project, we have extended COMPSs and PyCOMPSs to support the functionalities required by the Pillar Workflows and the interfaces for the integration with HPC, HPDA and AI computations. During the first reporting period of the project, we implemented two new decorators to support new types of tasks. First, the @http decorator is used to include invocations of rest services within a PyCOMPSs workflow. It allows creating workflows that combine service invocation with other computations supported by PyCOMPSs. This functionality is used by the UCIS4EQ workflow of Pillar III. Second, the @mpmd_mpi decorator is used to support “multiple program multiple data” MPI executions; this functionality is required by Pillar II workflows. Finally, we also implemented the @software decorator to support generic, sharable and reusable descriptions of software invocations.

In the second reporting period, we have extended PyCOMPSs in several ways: first, we have extended the container support for MPI and MPMD_MPI tasks. We have also extended the @software decorator to include the execution parameter description, which originally was managed with a separate decorator (@task). The support for data transformations has been included by implementing the @dt decorator. Moreover, we have integrated COMPSs with ParSODA and Ophidia. In the first case, the ParSODA computations are now generating PyCOMPSs tasks and its executions are managed by the COMPSs runtime. In the case of Ophidia, the Ophidia services are started/stopped as part of the COMPSs application using prolog/epilog phases, in such a way that Ophidia calculations can be included as part of the PyCOMPSs workflows.

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 first reporting period dataClay was extended with a functionality that enables efficient iteration of distributed datasets for their processing. The functionality is called split, and it provides a transparent way to get data partitions (logical groups of blocks that are in the same location). With these partitions, performance is improved as the system produces fewer tasks, thus reducing both scheduling costs and invocation overhead. This functionality is integrated with PyCOMPSs, and has been designed and implemented in a generic way, so that it is useful for some of the use cases in eFlows4HPC, as well as in many other situations.

This first version of the split mechanism loses the original ordering of the collection, which is relevant for some algorithms. Thus, during the second reporting period we improved the functionality so that it can provide the application with information about the positions of the elements, when needed. Also during this second period, the functionality was integrated with several dislib kernels as well as other popular data analytics algorithms.

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.

During the second reporting period of the project, we have identified in the use case the requirement of supporting synchronized access to Hecuba between producers and consumers. For this reason, we have designed and implemented into Hecuba a mechanism that provides this synchronization, supporting in this way online data analysis.

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.

During the first reporting period of the eFlows4HPC project, we extended the capabilities to load ds-arrays from different types of data. We added the support to load from hdf5 files and also from future numpy objects (Numpy objects generated by previous PyCOMPSs tasks). This feature allows

developers to combine dislib computations with other computations in the workflow without requiring synchronization of data after every step. During this period, we also developed some dislib methods, being the more relevant a blocked QR decomposition with three modes: full, economic, and r.

During the second reporting period of the project we have implemented two main extensions: first, it has been extended to support computations in GPUS; second, the dislib's estimator-based interface to support several distributed training approaches with different synchronization modes has been implemented, which enables the integration with EDDL and PyTorch to support deep-learning. In addition, we have developed some new methods such as a parallel implementation of the Tall and Skinny QR (TSQR) decomposition, an extension of the Grid Search to support the exploration parameters-sweep of HPC simulations, and a new KNN classifier.

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 second reporting period of the project, EDDL has been improved in the following directions, providing support to the different pillars where EDDL is being used:

- Autostop capabilities have been added. With this, a neural network training process can keep the best neural network model and stop the process once a model has been optimized (no further training process improves the model).
- The distributed training has been optimized with new strategies for batch distribution and synchronization. A parameterized distributed training process can now dynamically update the batch size in order to reach a certain/target network overhead.
- The Python library has been extended with a hyperparameter search mechanism. With this, a search method of optimized hyperparameters is now possible within EDDL.
- The ADAM optimizer within EDDL has been adapted in order to obtain better results.
- Several data formats required within the project have been added to EDDL.

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. The goal of Heat is to fill the gap between data analytics and machine learning libraries with a strong focus on single-node performance, and traditional high-performance computing (HPC). Heat's generic Python-first programming interface integrates seamlessly with the existing data science ecosystem and makes it as effortless as using numpy to write scalable scientific and data science applications.

Pillars to define the use cases and specific requirements. The first prototype implementation of the required functionality was also carried out. During the second reporting period of the project, we have focused on expanding our functionalities in distributed linear algebra and matrix decomposition, with a now fully functioning parallel Lanczos decomposition, and a draft hierarchical SVD functionality. Basic memory-distributed sparse operations are now available for the Compressed Sparse Row format.

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 Online 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 include:

- For the first release:
  - A new parallel operator to support loading of data from multiple NetCDF files concurrently in a single datacube structure;
  - Changes to the Ophidia internal import procedures to allow management of non-spatial oriented data (e.g., Tsunami scenarios).
- For the second release:
  - A new in-memory functionality to compute the climatological mean (i.e., the average values over the same time period on multiple years) to speed-up and ease the computation of extreme events indices;
  - A new parallel operator to support intercomparison between multiple cubes (i.e., more than two);
  - Extensions to the deployment schema for the integration with PyCOMPSs in order to use the same computing nodes;
  - Improved version of the functionalities developed for the first release.

The full set of changes is reported in the HISTORY file 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 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.

During the first reporting period of the project, the ParSoDA library has been ported to Python and it has been extended to support multiple execution runtimes. Specifically, according to the bridge design pattern, we defined the ParsodaDriver interface (i.e., the implementer of the bridge pattern) that allows the developer to implement adapters for different execution systems. A valid instance of ParsodaDriver must invoke some function that exploits some parallel pattern, such as Map, Filter, ReduceByKey and SortByKey. The SocialDataApp class is the abstraction of the bridge pattern and is designed to use these parallel patterns efficiently for running ParSoDA applications. It is worth noting that the execution flow of an application remains unchanged even by changing the execution runtime, which makes the porting of a ParSoDA application to new execution runtimes.

In particular, we included four execution drivers into ParSoDA-Python:

- ParsodaSingleCoreDriver, a driver that implements parallel patterns as simple sequential algorithms to be run on a single core, on the local machine. It is useful for verifying the correctness of a new ParSoDA Driver during its construction.
- ParsodaMultiCoreDriver, which runs the application in parallel on multiple cores, on the local machine, using Python's thread pools.
- ParsodaPySparkDriver, which runs the application on a Spark cluster. It is based on the PySpark library and requires the initialization of a SparkConf object.
- ParsodaPyCompssDriver, which runs the application on a COMPSs cluster. It relies on the PyCOMPSs binding to gain access to the COMPSs runtime.

In the second reporting period of the project, we focused on improving the ParSoDA integration with PyCOMPSs by adapting the execution workflow in order to optimize the generation of parallel tasks to be executed on the COMPSs runtime. In addition, we developed some study case applications for evaluating the performance of the library in terms of execution time and scalability.

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
- ParSoDA - Parallel Social Data Analytics
- 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
- 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