D2.2 Technology Evaluation, Containerization and Optimization Strategy

Version 1.0

Documentation Information

Contract Number	9555558
Project Website	www.eFlows4HPC.eu
Contractual Deadline	31.10.2021
Dissemination Level	PU
Nature	R
Author	François Exertier (Atos)
Contributors	Rosa M. Badia (BSC), Yolanda Becerra (BSC), Alessandro Danca (CMCC), Jorge Ejarque (BSC), Donatello Elia (CMCC), François Exertier (Atos), Jose Flich (UPV), Fabrizio Marozzo (DTOK), Anna Queralt (BSC)
Reviewer	Enrique S. Quintana Ortí (UPV)
Keywords	Optimization, technology, workflow, storage, container, Data Analytics, Machine Learning, HPC

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	Initial version for internal review
V0.2	Applied corrections related to internal review
V1.0	Version formatted for submission

Table of Contents

1. Executive Summary

This document presents strategies and methodologies that will be adopted in the eFlows4HPC project to select, adapt, integrate and optimize the different technologies that are proposed to be used for building the Workflows as a Service management stack, as well as the workflows themselves. It results from the studies conducted in three tasks of the WP2:

- Data analytics, machine learning libraries and workflow system optimizations
- Deployment optimization through container methodology
- Storage technologies

Based on the requirements issued from Pillars use cases, from components to be deployed on computing infrastructure, and from HPC (High Performance Computing) Data Centres constraints, a set of strategies is derived in order to optimize and facilitate the deployment and execution of scientific workflows by the eFlows4HPC platform.

Regarding libraries usage optimization, the result is to first focus on a specific problem related to data partitioning strategy. Workflow execution will be optimized by leveraging parallel execution and dynamic resource management. Container technologies will be used for applicative workflows management, taking care of containerized components availability and of related usage constraints. Strategy for storage is to use alternative solutions to traditional file systems, in order to optimize data operations within a distributed execution context; this is done in a per use case context.

Section 2 provides more information about the context and goals of this document, then the study is split into three main parts: section 3 about the optimisation strategies related to all libraries and runtime to be used for scientific workflows execution; section 4 about deployment optimisation through container technologies usage; and section 5 about optimisation strategies regarding storage technologies. The conclusion summarizes the results obtained from the previous sections, explaining how they will contribute to workflow optimization and have impact on the metrics established for the project.

2. Introduction

The eFlows4HPC project aims to deliver a “workflow as a service” platform that will enable the integration of HPC simulation and modelling with big data analytics (DA) and machine learning (ML) in scientific and industrial applications. The integration of these different technologies (HPC, ML, DA) in a single workflow raises complex issues related to the development, deployment and execution steps, and requires a complex lifecycle management.

The project introduces the HPC Workflow as a Service (HPCWaaS) concept, implemented on top of a software stack, the goal of which is to hide the complexity of a HPC/DA/ML Workflow execution to end users. The project requirements are issued from three application Pillars with high industrial and social relevance: manufacturing, climate and urgent computing for natural hazards.

As explained in deliverable D1.1[1], the eFlows4HPC software stack is composed of a set of software components:

- First, a set of repositories, catalogues, and registries contain the components that will be used within the workflows. It includes their core software components such as HPC libraries and DA/ML frameworks (Software Catalogue), data sources (Data Catalogue), ML models (Model Repository), workflows itself (Workflow Registry).
- Then, some components provide the functionalities to define, deploy and execute workflows by themselves; i.e., how the software components that are used within the workflows are deployed on the infrastructure, how required data is staged in or out through data logistic pipeline setup, how the workflows are executed, both at the higher level through a TOSCA based orchestration (YSTIA/Yorc) and at the lower level, through distributed tasks execution (PyCOMPSs). These components orchestrate the deployment and coordinate execution of the workflow components in federated computing infrastructures.

The combination of all these technologies raises important optimisation issues, both at the DA and ML libraries level, at the data management (storage) level, and at the workflow execution level.

The goal of this document is to gather the results of the studies which have been conducted to optimize the usage of DA/ML libraries, of the storage technologies, and of the workflow orchestration systems. How the container technologies could facilitate the usage of these components has also been addressed.

One of the goals of the project is to improve the performance and reduce the energy required to execute complex workflows. This reduction in energy is tackled by multiple activities in the project. By automating the development, deployment, and execution of workflows, besides widening the access to HPC systems to existing and new user communities, we will be reducing the cost of using such resources. This cost reduction will directly impact the time required from users/developers, and indirectly will also imply a more efficient use of the HPC resources and therefore a reduction in energy required to operate them.

The optimization in orchestration that will be tackled in the project, enabling the overlap of different types of computations and the use of better resource management, will also benefit the energy consumption of the workflows.

Another contribution of the project to energy efficiency will be provided by the activities in WP3, which aim to optimize specific kernels for specialized architectures such as GPS, FPGAs and the EPI.

3. Data analytics, machine learning libraries and workflow system optimizations

This section presents the results of task 2.1 of WP2, "*Data analytics, machine learning libraries and workflow system optimizations*", at this stage of the project. The goal of this task is to develop the required optimizations for obtaining a performant, scalable and energy efficient platform. Specifically, the optimizations are particularly designed for the different DA components, for ML libraries, and for workflow runtimes/orchestrator engines used in the project.

In general, we deal with an *optimization problem* whose ultimate goal is to minimize a cost function (or equivalently maximize a utility function). Optimization problems are formalized through quantitative techniques and mathematical models, characterized by a set of constraints, an

objective-function and the unknown decision variables to be determined. The search for solutions of an optimization problem is carried out through specific logical-mathematical procedures (i.e., optimization algorithms). There is a wide range of real-world problems in several fields that can be formalized as optimization problems, such as production planning, investment and portfolio management, logistics and transport management, etc.

Within the project, we focus on the optimization of software components, which have been grouped into three main groups: ***data analytics, machine learning libraries and workflow runtimes/orchestrator engines***. For each of them, examples of possible optimizations are provided.

Data analytics. This group includes all software components that allow data to be processed within an end-to-end knowledge discovery process, from raw data to the visual interpretation of the extracted patterns. This group includes *Ophidia*, *ParSoda* and *DMCF* that, in different ways and on different types of data, allow designers to pre-process data, prepare them for the analysis, analyse them using machine learning/data mining algorithms and, finally, visualize the results. Possible optimizations in this group for example include the following aspects:

i. the way in which data is prepared, moved and modified from one phase to another of the analysis process;
ii. how data are partitioned in order to be analysed efficiently, reducing overhead while ensuring a good level of parallelism;
iii. the choice of a suitable set of parameters for configuring the different algorithms, which involves a study on how these parameters can affect the quality of the results and the execution times;
iv. the possibility of having new algorithms and tools available within the different softwares for coping with specific analyses.

Machine learning libraries. This group includes a series of software libraries that allow to perform advanced data analysis through the application of ML techniques. It includes *Dislib*, *EDDL* and *HeAT*, a collection of methods and algorithms in the field of ML that can improve automatically through experience by leveraging a large set of training data. Some optimizations in this group include:

i. tools to transform input data into a suitable format for the different libraries;
ii. proper data chunking strategies for analysing data in a parallel and efficient way;
iii. techniques for hyperparameter-tuning aimed at properly setting the different algorithms;
iv. the implementation of ML algorithms or tuning strategies which are not yet available in a library.

Workflow runtimes/orchestrator engines. This last group includes two software frameworks (*PyCOMPSs* and *Yorc*) used to orchestrate the execution of different workflow computations, which can include just a simple task or sub-workflows implemented with different HPC libraries or DA/ML frameworks. The possible optimizations in this group refers to the following aspects:

i. coordination of all operations that are needed to manage the interaction with all the software systems and libraries exploited in the workflow;

ii. choice of a suitable strategy for data partitioning and storage;
iii. assignment of the different tasks to computing resources so as to minimize the movement of data between nodes;
iv. optimize workflow orchestration, allowing the simultaneous execution of tasks, in order to maximize throughput, improving the exploitation of resources and consequently the energy efficiency of the overall execution.
v. optimization of memory usage, in order to maximize in-memory computation, avoiding, at the same time, problems related to memory saturation.

Traditionally, these phases have been executed as a static pipeline where only one of the phases is executed at a time. In each step, a specific runtime is in charge of managing the execution of the processes in the available resources. However, nothing prevents that, introducing advanced workflow and resource management techniques, all the phases can be dynamically executed at the same time allowing overlapping of the different phases, speeding up the overall execution and performing better resource management and consequently improving the energy efficiency of the overall execution.

Among all possible optimizations, in the following we focus on two specific problems: **data partitioning** (or *data chunking*) and **workflow orchestration**. We firstly introduce the problems along with some implications in the context of distributed computation environments. Afterwards, we propose solutions referring to specific tools / software present in the project.

3.1 Data partitioning optimization

Data partitioning refers to splitting a dataset into small and fixed-length units called *chunks*, in order to enable data-parallel processing and storage in distributed systems. There are several issues related to data partitioning, mainly related to *load balancing* and *parallelism*.

When distributing a dataset on a set of nodes within a distributed computation environment, the choice of the destination node for a given chunk, i.e. the node where that chunk will be stored, is often crucial. System load should be balanced over the distributed system nodes to improve overall performance, utilization of resources, response time and stability. For this purpose, an optimal distribution of the chunks can avoid excessive communication, caused by the migration of the different chunks from heavily utilized nodes to underloaded ones. Moreover, the choice of the destination node can be guided by its hardware characteristics, which should be in line with what is needed for processing the chunk.

Regarding parallelism, the choice of the size of the chunk can heavily affect the trade-off between single node efficiency and parallelism. Specifically, a larger size reduces parallelism (less chunks) but makes tasks larger. Although this can lead to an overhead reduction, it must be ensured that the size of the chunk does not exceed the memory available on the individual nodes, so as to avoid memory saturation errors. On the other hand, a smaller size leads to a finer exploitation of parallelism, while introducing larger overhead due to communication and synchronization, which can have a negative impact on performance.

Focusing on this last aspect, we can formally define the choice of the correct chunk size as a *fixed-size chunking* problem[2]. Specifically, given a dataset D characterized by n rows and m columns, we must determine the size of the chunk $S = (n', m'), n' \leq n, m' \leq m$ which leads to the best partitioning of D on the distributed system, matching the optimal value for performance in the efficiency-parallelism trade-off.

3.1.1 A machine-learning approach for chunk size prediction

The problem of estimating a proper size for data chunks in data-parallel applications is of great interest as it can heavily affect system performance and scalability.

Our solution treats this problem from the point of view of **regression**. In particular, our approach is comprised of the following steps:

1. Prepare a set of executions to find the optimal chunk size to reduce/optimize the execution time. This step is characterized by several degrees of freedom, as different algorithms must be taken into account, as well as a wide range of configuration, input data and hardware characteristics. In such a vast search space, we need to find an effective exploration strategy in order to determine a sub-optimal value for the chunk size. For this purpose, we can compare several techniques used for hyperparameter optimization, including *grid search*, *randomized search* and *genetic algorithms*. At the end of this step we obtain a dataset in which each row represents an execution described by:
    - **Algorithm-** and **infrastructure-related features**, along with dataset characteristics.
    - **The optimal chunk size** estimated previously (i.e., the *target variable*).
2. Train a regressor to learn the patterns that link execution characteristics and the relative chunk size. The output is a regression model able to estimate the optimal chunk size for a given task based on its features, input data characteristics and the underlying system infrastructure.

This approach has already been successfully leveraged to improve the in-memory execution of data-intensive workflows on parallel machines, showing its effectiveness with Apache Spark used as a testbed[3].

3.1.2 Use case: Dislib

The aforementioned technique, especially designed to address the problem of the choice of a proper chunk size, can be applied to a wide range of frameworks for distributed data processing. In fact, the majority of the state-of-the-art systems, such as *Hadoop*[4], *PyCOMPSs*[5] and *Dislib*[6], leverage a data-parallel approach that involves a data partitioning step for distributing the dataset across the working nodes. Consequently, the application of the solution described above can lead to a performance improvement by optimizing the partitioning process, which in turn reduces overhead while ensuring a good level of parallelism and throughput.

In particular, among the main frameworks and libraries of interest for distributed computing, we selected ***Dislib*** as a testbed. This is a distributed computing library built on top of PyCOMPSs that provides distributed mathematical and machine learning algorithms through an easy-to-use interface. It is inspired by *scikit-learn* and *numpy,* and comprises various machine learning algorithms, such as *K-means*, *DBSCAN*, *Support vector machines* and *Random forests*. The main concepts around *Dislib* are:

- *Data handling*: load data from files in common formats, for example CSV and LibSVM.
- *Unified interface*: *Dislib* provides a *scikit-learn* inspired interface for the different algorithms (i.e., fit, predict, etc.), which makes it easy to adopt, allowing also an easy transition of existing code from scikit-learn to *Dislib*.

- *Interoperability*: *Dislib* can be easily integrated into any existing PyCOMPSs application, and can run in any computing platform supported by PyCOMPSs as, e.g., clouds and clusters.
- *Distributed arrays (ds-array)*: a built-in 2-dimensional array that can be operated in parallel, which is used as the main input for the different algorithms. Distributed arrays are divided into blocks that are stored remotely.

In this framework, the degree of parallelism is controlled by the **ds-array's block-size**, which defines the number of rows and columns of each block. Therefore, the choice of the optimal block size is essential in order to exploit the full potential of *Dislib*, hence the possibility of effectively applying the proposed solution.

3.2 Workflow orchestration optimization

The execution of complex workflows tackled by the eFlows4HPC project combines the execution of HPC simulations together with DA algorithms and ML training and inference. Traditionally, these phases have been executed as a static pipeline where only one of these phases is executed at a time. In each step, a specific runtime is in charge of managing the execution of the processes with the available resources. However, nothing prevents that, via the introduction of advanced workflow and resource management techniques, all the phases can be dynamically executed at the same time speeding up the overall execution, and attaining better resource management and consequently improving the energy efficiency of the overall execution. An example of this fact is shown in the execution trace in Figure 1, where the upper side shows an execution by phases and the bottom shows an execution overlapping the phases.

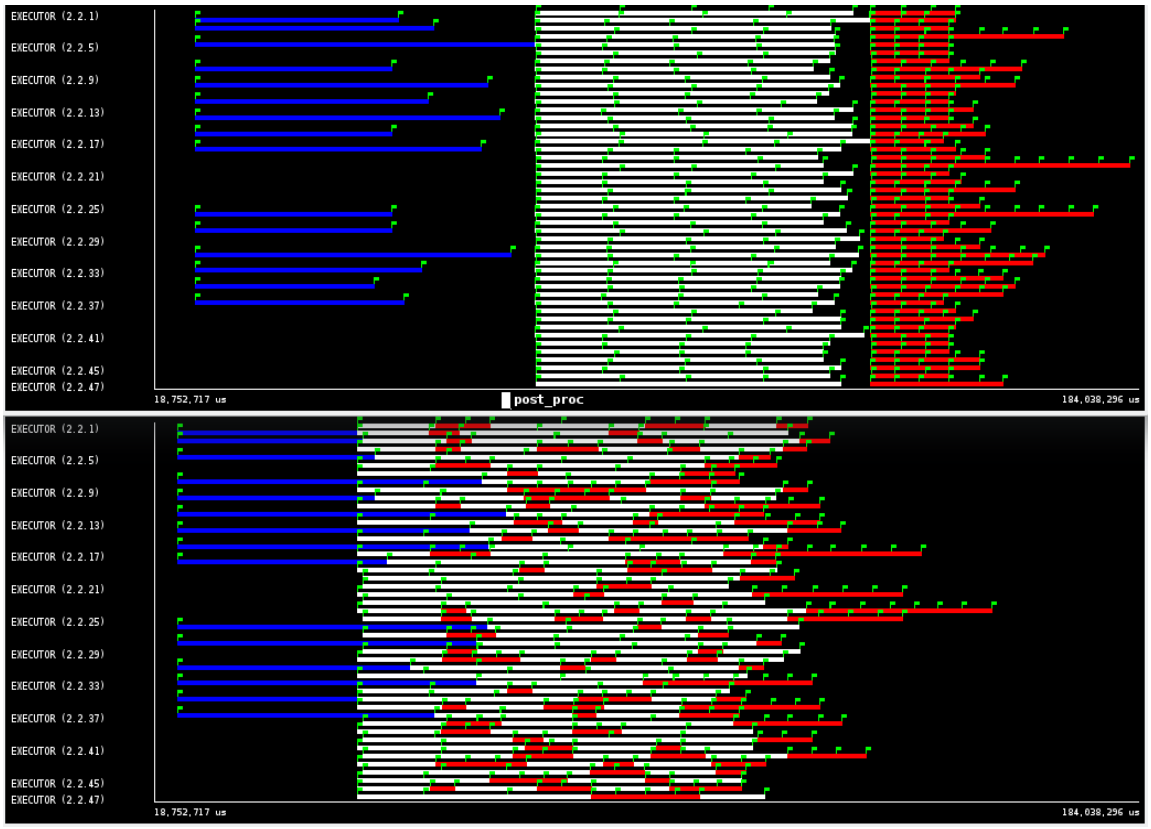

Figure 1 Comparison of workflow execution by phases (top) and overlapped (bottom)

The strategy to optimize the workflow execution aims at using information extracted at runtime from the workflow definition in two terms. From one side: orchestrate the execution of the different HPC/DA/ML framework runtimes, and from the other, adapt the resources assigned to the workflows according to the real application parallelism.

3.2.1 Runtimes Coordination

When running workflows combining different HPC/DA/ML phases, it is important to control the resources used by the different frameworks in order to balance the load across all the available resources. Configuring the resources used by the HPC/DA/ML frameworks can be done either by a configuration file provided at startup, a management API (Application Programming Interface), or by limiting the resources from the OS (Operating System). In eFlows4HPC, this orchestration will be transparently done by the COMPSs runtime. Framework executions can be defined as special tasks defined in a PyCOMPSs workflow. The COMPSs runtime is able to detect the data dependencies between the different executions, inferring which ones can run in parallel. Then, they can be scheduled in the available resources and execute and configure the required framework according to the allocated resources.

3.2.2 Adaptive Resource Management

Another important implication of the dynamic execution of the complex workflows is that the parallel workload varies during the workflow execution. The static resource management provided by traditional job schedulers does not match with this dynamic workload and can produce two effects:

- In the embarrassingly parallel phases, the execution time slows down because the number of parallel processes is limited by the requested resources.
- In the reduction phases, the parallel workload decreases so some resources become idle.

A better resource management could be performed if we are able to predict the current parallel workload of the application; compare it with the current resource capabilities; request more resources when the parallel workload is larger than the current resource capabilities; and release resources when resources become idle. The COMPSs runtime has the information to infer this knowledge. It can use the information to estimate the parallel workload according to the computing requirements of all dependency-free tasks (running or pending) and it also knows the available resources to execute the workflows. With a simple comparison, it can easily decide when to contact the resource manager to request or release resources.

3.2.3 Energy consumption implications

The techniques to improve the workflow orchestration presented in the previous sections mainly target the reduction in time of resources in an idle state. Therefore, they have a direct implication on the energy consumption. If we improve the execution time by just doing a better usage of the resources, the energy consumption will also be reduced.

However, other techniques to improve the total execution time are not always translated to energy savings. For instance, increasing the parallelism of an application can considerably reduce

the execution time, but it is very likely that the energy consumed during the execution is increased. Increasing the parallelism of an application is normally achieved by applying a finer grain data partition to enable the usage of a larger number of resources. However, it increases the communication and management overhead and reduces the parallel efficiency of the application. For that reason, it is also important to find an optimal data partition to achieve the desired execution time without increasing the energy consumption.

For example, total execution time can be reduced in neural network training processes. Indeed, with some of the NN libraries (e.g., EDDL) the training process of a neural network can be distributed over the system resources (e.g., using GPUs). With the data parallelism approach, the model is replicated on each GPU and a subset of the training data is used by each GPU. During the training process, all GPUs synchronize their models, thus, converging on the same single model. The way synchronization is performed and the frequency of synchronization leads to a trade-off between training time and accuracy of the model, but can also impact energy consumption.

4. Container based Deployment

This section presents the results of task 2.2 of WP2, **Deployment optimization through container methodology,** at this stage of the project. The objective of this task is to study and implement the way container technologies will be used to speed-up and improve the overall flexibility of the deployment process, applied both to the project software stack (HPCWaaS and all related software components) and to the applicative workflows. Therefore this is strongly tied to the work conducted in task 1.5 of WP1, **Widening access to HPC systems: HPC Workflow as a Service (HPCWaaS)**, the goal of which is to provide the HPCWaaS. The HPCWaaS platform will enable the deployment and execution of complex workflows by providing the *developer* with a means to define, register, and deploy workflows, and the *end user* with facilities to deploy and execute such registered workflows.

The rest of this section is divided into four parts:

1. Review of the requirements to be addressed by the application of these container technologies
2. Container technology to deploy the eFlows4HPC software stack
3. Container technology used by the eFlows4HPC service (HPCWaaS) to deploy the workflows
4. Container technology usage constraints

The first part lists the project requirements addressed by task 2.2.

The second part lists the components of the eFlows4HPC software stack and describes if they can be deployed through a container technology.

The third part presents the eFlows4HPC components in charge of workflows deployment, and how they may rely on container technology to deploy workflows in the form of container images.

Finally, the fourth part of the study describes the constraints related to the usage of container technology within the context of the project, both from the Data Centres point of view (security issues...) and from the components and pillars point of view (availability of container images, container-based components...).

4.1. Project and Pillars requirements addressed by Container Technology Concerns

The main requirement regarding the usage of container technologies in the project is to accelerate the deployment and enhance the flexibility of the workflows. Among the requirements listed by the pillars (see deliverables D4.1, D5.1, D6.1), and more globally at the project level (see deliverable D1.1), some should be considered with particular attention.

Table 1: Requirements related to Container Technology usage

P3-6	Portability	Workflow components must be portable to several infrastructures
CMP-1	Access to HPC specific devices	Workflows developed with eFlows4HPC stack must be able to access the specific HPC hardware such as High Performance networks, accelerators or special CPU vectorial instructions.
CMP-2	Support Optimized kernels	Workflows developed with eFlows4HPC stack must be able to support the architecture-optimized kernels and libraries
CMP-3	Service deployments	The eFlows4HPC software stack should support the deployment of Data Bases and Services required by the DA and ML frameworks in auxiliary cloud and HPC centers
HPC-3	Singularity Container support	The usage of containers in the HPC system must be compatible with singularity containers

Using container technology is of course a way to ensure a certain level of portability among distinct infrastructures (P3-6), as several Data Centres or Clouds may be considered to deploy a same workflow. However, this does not address all portability issues, and special attention should be paid to some specificities encountered when executing HPC simulation, ML or DA codes. Actually, using containerized code should not prevent from taking into account some specific HPC hardware (CMP-1). There are some optimized ML or DA libraries that should be used within the container images (CMP-2).

The eFlows4HPC workflow deployment model is hybrid, in the sense that some steps are deployed on HPC clusters for job-like execution, while other steps address the deployment of services (e.g., a storage system), both should be considered when applying a container technology (CMP-3).

Finally, most HPC Centres do not allow Docker technology to be used on the HPC clusters. Therefore the alternative container technology Singularity should be supported in these cases (HPC-3).

4.2. Deploying the eFlows4HPC software stack through Container Technologies

The table below lists all the software components to be used in the project and provides information about the availability of a container-based deployment. In case this is not yet available, alternative available distributions are provided. The list and description of these components can be retrieved from deliverable D1.1 "Requirements, metrics and architecture design"[1]. These components can be categorized in the following types according to their role in the workflow lifecycle and the expected deployment:

- Infrastructure Components: they are services and tools to manage the workflow lifecycle but are installed outside of the computing infrastructure
- Runtime Components, which are components that must be deployed together with the workflow in the computing infrastructure
- Software Components: they are ML/DA frameworks and HPC libraries that are the specific software required by the workflows and must be deployed in the computing infrastructure

Table 2: Software Components

Component	Description	Container-based distribution	Comments
Infrastructure Components			
Model Repository	Model repository based on MLFlow	Yes	
FastML	Model Management and Training	Yes (Docker)	
Ystia suite	This provides the Alien4Cloud (TOSCA GUI) Ystia Forge (TOSCA repository) and Yorc (TOSCA Orchestrator)	Yes (Docker)	
Data Logistic Service	Based on Apache AirFlow	Yes (Docker compose)	
Data Catalog	Registry of the data-sets involved in the workflow	Yes (Docker)	
Unicore	Federated access to HPC	No	CentOS Debian packages

Runtime Components			
PyCOMPSs	Task based programming model for building workflows from python scripts	Yes (Docker/Singularity)	
dataClay	Distributed Object Store	Yes (Docker/Singularity)	
HECUBA	Key-value Data Store	Yes (Docker/Singularity)	
DA/ML Frameworks			
Ophidia	HPDA	In progress.	Currently, RPM or DEB packages are available. Potential issue with the MPI
EDDL	Neural Networks	Yes (Docker)	Potential issue with MPI and GPU parts and architecture-optimized libraries (Linear Algebra)
Dislib	Distributed Machine Learning algorithms	Yes (Docker)	Potential issue with architecture- optimized libraries (Linear Algebra)
HeAT	Data Analytics and Machine Library	No.	Pip installer available Potential issue with MPI and GPU parts and architecture-optimized libraries (Linear Algebra)
ParSoDA	Data Analytics Java library	Yes (Docker)	
DMCF	Data Analysis Workflows	N/A	Current version is based on MS Azure Cloud software. Each element is a MS Azure component
HPC Software			
Kratos	Parallel Multiphysics simulation software	No	Cmake and pip installer Potential issue with MPI parts and architecture-optimized libraries (Linear Algebra)

ParMMG	Software for parallel mesh adaptation of 3D volume meshes	No	Cmake installer
Salvus	Earthquake simulation software	No	Potential issue with MPI parts
Tsunami-HySEA	Tsunami simulation software	No	Potential issue with MPI and GPU parts
FESOM2	Finite Element Sea Ice-Ocean Model	Yes (Docker)	Cmake installer Potential issue with MPI parts
OpenIFS	Numerical Weather Prediction Model	No	Potential issue with MPI parts
OASIS3-MCT	Model Coupling toolkit	No	autotools (configure, make)
CMCC-CM3	Climate Model	NA	No portable

This shows that most of the components to be used in the project are or will be available as container images. Moreover, for all components to be deployed via TOSCA/Yorc, it would be possible to create Docker files to build Docker images to be deployed by the orchestrator.

This means that the strategy to adopt will be twofold:

- As most of the HPCWaaS software stack will be available through container images, we consider building the stack itself as a container-based distribution, which will facilitate its installation and the adoption of the eFlows4HPC framework and
- As the components provided by eFlows4HPC, to be used within the applicative workflows (such as DA libraries, ML frameworks…), will also be available as container images, it should be studied how the deployment of workflows through container technologies could be facilitated. This is described in more detail in the next section.

4.3. Container technology used by the eFlows4HPC service to deploy the workflows

Workflows are managed within eFlows4HPC at two levels: the higher level is handled by the TOSCA-based YSTIA/YORC orchestrator, while the lower level is handled by PyCOMPSs. Regarding ML workflows, FastML provides advanced features and APIs to deploy “training” workflows. These three aspects of workflow management are addressed in the next subsections.

4.3.1. Yorc High Level Workflows Management

Yorc is a TOSCA-based orchestrator that manages application lifecycle over a hybrid infrastructure. It takes care of resource allocation, software provisioning, application start/stop, and undeploying, as well as workflow execution. Regarding containers, two features are available today:

- Deploying Slurm jobs on Singularity
- Deploying Docker based application

These capabilities are subject to evolve and/or be complemented according to the eFlows4HPC requirements.

Yorc will be used for the management of high-level workflows of HPCWaaS. This is being defined in task T1.5. At the higher level, workflows will be defined using TOSCA, specifying the software components to be used and the data flows. This means that Yorc will be used for defining, deploying and executing the applicative workflows.

The strategy will be

- To use the Yorc capability to launch jobs running in Singularity containers for all the workflow components to be executed on an HPC infrastructure
- To use Yorc capability to deploy Docker-based application for all the workflow components to be executed on the Cloud infrastructure
- In case, for some reason, containers cannot be used, it will still be possible to use Yorc ability to deploy non containerized components (e.g., through Ansible…).

Although this still needs further study to be conducted in task 1.5, applying this strategy will facilitate the HPCWaaS workflow implementation, as for example the Software Catalog may be implemented through one or several container registries while provisioning a software component could be a simple image pull.

4.3.2. PyCOMPSs Low Level Workflows Management

As previously mentioned, PyCOMPSs is a task-based programming model for programming parallel workflows from Python scripts allowing the integration of different software in a single workflow. PyCOMPSs already has a certain support for containers. PyCOMPSs is distributed as a docker image available in DockerHub[7] and it currently provides two modes of execution with containers:

- **Whole PyCOMPSs application deployed as containers**[8]: In this case, the image containing the PyCOMPSs runtime is extended with the software used in the workflow, and several containers are deployed to run the PyCOMPSs application: one container running the master processes; and the other running the worker processes executing the tasks. The advantage of this approach is that the overhead of the execution will be similar to that of a bare metal execution due to the containers being mainly initiated at the application startup. The main drawback of this option is that users need to create an image with all the application dependencies. This can be a complicated task depending on the experience of the end-user.
- **PyCOMPSs tasks are executed in containers**: In this mode, the PyCOMPSs runtime is installed on the bare metal and either a Python method or binary execution tasks (and potentially also the MPI tasks) can be annotated with *@container* decorator. Tasks

annotated with this decorator are executed in their corresponding container. This approach requires less work for the application developers because they can use already built container images for each piece of invoked software. In contrast, it requires a previous installation of the PyCOMPSs runtime and the overhead of running the containers is performed per task invocation.

To overcome the drawbacks for the aforementioned execution modes, two strategies can be followed:

- For the first mode, we could automate the PyCOMPSs application image using tools such as EasyBuild or Spack which can be leveraged to install HPC software inside container images.
- For the second mode, the container task support in PyCOMPSs can be extended to enable its execution from a containerized version of PyCOMPSs. In this case, a mechanism to access the container engines (singularity/docker) from the PyCOMPSs worker container must be designed and implemented.
- To reduce the execution overhead in this second mode, the deployment of container images in the infrastructure can be integrated in the installation workflow defined in the TOSCA part. This will increase the time of the installation phase but it will reduce the execution phase which is the time experienced by the end-user.

4.3.3. ML Workflows Deployment with FastML

FastML provides GUI (Graphical User Interface) / CLI (Command Line Interface) / REST API to manage ML workflows, in particular for AI (Artificial Intelligence) model training management. For this purpose, it currently relies on Docker images holding both the ML framework, the model and the training code, to deploy them through the Slurm job scheduler on the HPC infrastructure. Such containerized training code can be deployed either through a Docker or through a Singularity runtime.

Today, most of the ML/DL frameworks and models are provided as Container images, and many aspects of AI require the power of HPC infrastructure. Therefore, Data Scientists are faced with the complexity of both container technology and HPC infrastructure.

The strategy will be to use FastML for any part of a workflow that involves a ML step, in order to take advantage of the availability of ML components as Container images as well as to hide the complexity of using container technologies on top of HPC infrastructure to the user

4.4. Container technology usage constraints in HPC

The HPC parts of eFlows4HPC workflows will be deployed on HPC clusters. Most HPC centres rely on Singularity container technology for application deployment and do not enable Docker usage for security reasons. Singularity is a container technology comparable to Docker. However, it has always been the preferred choice for HPC usage for several reasons, the main ones being:

- Docker requires superuser privileges at many stages while Singularity uses the running user. This is most of the time not compliant with security rules of HPC centres which are shared by many users

- Docker requires a daemon to handle the container processes, while Singularity does not need it
- As it has mainly been designed and used for HPC, Singularity natively supports HPC Interconnects, OpenMPI… There is also an integration of Singularity with the main HPC schedulers like Slurm.

It has to be noticed that Docker images can be used with Singularity, thus opening access to the huge number of Docker images available.

Components like FastML and Yorc rely on container technology for handling job deployment and they are currently able to deploy such jobs in Docker images through the Singularity runtime.

Regarding the usage of containers in an HPC context, several aspects require special attention. Images created using the standard distributions available in public registries like Docker hub work in supercomputers if they have a compatible ISA (Instruction Set Architecture), such as amd64 or x86_64, and the linux kernel. However, depending on the application, the performance achieved with these images will not be close to the peak performance of the computer. Extracting good performance for HPC simulators within containers on HPC infrastructures requires some features enumerated in the following paragraphs.

- **Processor Optimizations:** It is common that these standard images are compiled in laptops with optimizations performed for common processors. However, if we want to get the full benefit of the processor capabilities, the compilation process must target the specific architecture optimizations (Intel AVX, ARM AFX64, etc.) provided via the compiler flags.
- **Efficient Networking:** A similar issue occurs for MPI. These images are compiled with libraries and drivers using a common networking protocol stack (TCP/IP/Ethernet). However, MPI is really efficient in HPC when using high-performance networks (such as Infiniband or OmniPath) with support for RDMA which considerably reduces the networking overhead. To enable this capability from containers, container engines must provide access to the specific network devices and their specific software stack (such as runtime libraries, drivers or kernel modules).
- **Access to accelerators:** Although new computers include some kind of accelerator, the software stack required for accessing this technology depends on the vendor and model of the accelerator. Images should be created according to the underlying accelerator in order to compile applications with the required libraries to manage the execution in that accelerator. Moreover, as in the MPI case, the container engine must provide access to the GPU devices and its specific software stack (such as runtime libraries, drivers or kernel modules).
- **Optimized Parallel Kernels:** Most of HPC applications rely on computational kernels (such as Linear Algebra operations) which are implemented by specific libraries. Standard OS distributions provide precompiled versions of these libraries, which work efficiently for sequential execution on small multi-core computers. However, they do not work well for large multi-core processors present in current HPC systems. Most HPC processor vendors (such as Intel, ARM or AMD) provide versions of these libraries, which are optimized, for large multicore processors improving the performance.

A consequence of all the aforementioned issues is that there is a trade-off between performance and portability. Extracting the best performance from a supercomputer requires generating images according to the underlying infrastructure. In consequence, this reduces the portability of

the container images because devices or processor capabilities available in one supercomputer are not available in another.

The strategy to optimize this trade-off and provide a good performance ensuring a certain portability can relies in two solutions:

1. **Library replacement at container engine.** Leveraging the Application Binary Interfaces, container images can be generated with standard OS distributions including the required libraries at the application level. Those libraries that are required to access a specific hardware or implement an application kernel can be replaced by their optimized versions installed in the host space. This is commonly used to provide access to efficient MPI implementations[9] and GPU drivers[10] [11] while keeping a certain image portability. This option is a good solution if the application performance is mainly attained via MPI, GPUs or an optimized library. However, it still requires having an image for the supercomputer ISA architecture.
2. **Image generation at deployment time.** A second solution is integrating the image building process in the workflow deployment process. Instead of specifying the container image, which contains the software, we could include a receipt for building the software in a base container image, and define a step in the deployment phase to build the container images required by the workflow according to the provided receipts and the features of the target supercomputer. There are solutions that partially cover the functionalities required to achieve this goal. One option is combining tools to automate the installation of software in HPC such as Spack or EasyBuild within the container image build process[12]. These tools facilitate the installation of HPC software taking into account architecture compilation optimizations and optimized libraries included in the most frequent HPC software stacks (compiler, MPI version, Linear Algebra libraries, etc.). The European Environment for Scientific Software Installations[13] goes a step further by proposing the combination of EasyBuild with the Archspec python library[14] to automate architecture optimized installation. The archspec library can extract architecture related features from the target node and pass them to the EasyBuild configuration in order to install software for a specific supercomputer. Despite they have not been designed to create container images, this methodology can be easily integrated with container image creation tools such as Buildah[15] or Docker Buildx,[16] which are used to facilitate the creation of OCI-compliant container images and multi-architecture builds. Information extracted from the target architecture could be used to generate the manifest files describing the image creation, replacing typical *deb* or *rpm* binary package installations by Easybuild or Spack installations with architecture optimizations.

5. Storage technologies

This section presents the results of task 2.3 of WP2, **Storage technologies,** at this stage of the project. The objective of this task is to optimize the storage-related aspects of the software stack, by replacing the traditional file systems with alternative storage solutions in order to improve the performance of the workflow.

Three alternative storage technologies, already introduced in D1.1[1], will be considered as an alternative to file-based storage: dataClay[17], Hecuba and Ophidia. Each of them will be integrated

in the appropriate parts of the Pillars' workflows in order to improve data access and the overall performance of the workflow.

dataClay is a distributed platform that manages data in the form of objects, enabling seamless access and storage of complex and possibly distributed data structures (matrices, lists, graphs, ...). The physical location and format of the data is transparent to the application developer. Noticeably, the objects stored in dataClay also include the methods that enable their manipulation, such as retrieving or updating the data they contain, or performing arbitrary computation on them. In this way, data locality is exploited as data does not need to be transferred to the application to be processed: instead, part of the processing happens within dataClay, and only the results are transmitted to the application, thus notably reducing the amount of data transferred.

Hecuba is a set of tools that provide programmers with transparent and efficient access to key-value databases. The current implementation can interact with Apache Cassandra. Hecuba code is organized into two different layers that can be used together or independently. The first layer implements a Python API that allows programmers to access the persistent data as regular in-memory Python objects. The second layer is implemented in C++ and contains the code to interact with the database. This code implements some optimizations in the access to the database as data prefetching, data caching and data locality enhancement. Hecuba is integrated with PyCOMPSs to enhance data locality and to avoid serialization when passing the data to task persistent objects as parameters.

Ophidia represents an HPDA framework for the analysis of large scientific multi-dimensional datasets, by joining HPC and Big Data paradigms[18]. It is worth mentioning that Ophidia provides the features to support both data analytics and management, through an internal, in-memory and distributed storage model to handle multi-dimensional scientific data (through the datacube abstraction). Even though the main focus is on parallel scientific data analytics its integrated storage layer represents an important aspect for managing large-scale datasets. Ophidia has been mainly used for climate sciences, even though it has also been exploited in other scientific domains.

The approach followed within this task has been to analyse the storage-related requirements of the Pillars, and also identify which are the steps that can benefit most from adopting these alternative storage solutions. In order to avoid overlapping efforts, after this analysis we have chosen the storage solution that best fits each subproblem, according to its characteristics, instead of considering different alternatives for the same (part of the) workflow. This analysis, and thus the potential optimizations to be applied, may be extended during the project, as we have started focusing on those parts of the workflows where the theoretical benefits are more evident.

As the advantages of applying alternative storage solutions are tightly coupled to the specific requirements and computations that the workflows perform, the remainder of this section is divided into the different Pillars. For each of them, we explain the requirements that will be addressed, extracted from D1.1 (general)[1], D4.1 (for Pillar 1)[19], D5.1 (for Pillar 2)[20], and D6.1 (for Pillar 3)[21]. We also justify which are the storage solutions initially chosen to address these requirements, and explain the potential benefits that we aim to obtain with their integration in the workflow.

5.1. Pillar I: Manufacturing

As stated in D4.1, the overall goal of Pillar I is to provide an integrated workflow enabling the development of Reduced Order Models (ROM) to be leveraged in order to derive Digital Twins for manufacturing applications. The underlying objective is to enable the effective usage of large scale HPC hardware in speeding up the generation of ROM and to enable the solution of problems larger than it was possible prior to the implementation of the new capabilities.

In the following subsections we will first summarize the requirements that are relevant to possible storage optimizations, and then explain the proposed optimizations and how they will be addressed in eFlows4HPC.

5.1.1 Requirements related to storage

Among the set of requirements related to Pillar I (reported in D1.1[1] and D4.1[19]), the following table summarizes those related to storage:

Table 3: Pillar I Requirements related to Storage

P1-1	Distributed SVD	Requires an optimized distributed execution of the randomized Singular Value Decomposition (SVD) algorithm to analyze large scale matrices which can exceed the memory of a single cluster node
P1-2	Storing of hyper-reduced model	Requires storing and transferring the meshes and the trained ML model needed to reconstruct the hyper-reduced model, together with the solver executable needed to run it.
P1-4	Clustering model	Clustering algorithms as an option to improve the reduction ratio. Here both training data and the output to be used in the inference step need to be saved.
P1-5	Persistent storage	Requires persistent storage for data to be consumed between the steps.

Additionally, Table 4 in D4.1 lists a set of data requirements, which define the kind of data structures produced/consumed at each step of the workflow. Some of these data structures (namely the *simulations definitions* and the *mesh*) are the initial inputs of the workflow, and they are obtained from files following specific formats, such as JSON, HDF5, or other solver-specific formats. These files are read and then mapped to intermediate data structures, in order to support the computations that need to be performed. All the intermediate data in the workflow, namely the *snapshot matrix*, the *reduced model*, and the *intermediate matrices for hyper-reduced model*, is represented in the form of matrices. Finally, the output results of the workflow will be provided either in HDF5 or in a native Machine Learning format.

5.1.2 Proposed optimizations in eFlows4HPC

As stated above, important goals of this Pillar are increasing performance in the generation of ROM, as well as increasing the size of the problems that can be solved. These goals refer to the intermediate building blocks of the workflow (blocks 2 and 3 from Table 1 in D4.1), which are the most computationally intensive. Thus, the goal in eFlows4HPC for storage would be to provide a more efficient way to deal with the data managed during these computations, in order to optimize performance.

The aim of those blocks is to compute the SVD on large scale matrices, possibly exceeding the memory of computing nodes in a cluster. As the SVD algorithm is part of Dislib, which is based on PyCOMPSs, using this implementation will provide the required distribution of the algorithm, which also implies the distribution of the matrices involved in the calculation. The main data structure managed by Dislib is the *ds_array*, which represents a two dimensional array (a matrix), organized into blocks (submatrices) to enable distribution and parallelization of the computation by means of PyCOMPSs.

Given the characteristics of this computation, and the features of the storage solutions considered, the storage solution chosen to optimize this computation is dataClay. dataClay is a distributed data store able to manage data in the form of objects, which can represent any object-oriented data structure. It allows Python applications (or libraries, such as Dislib) to transparently store their objects, and it is able to execute arbitrary code associated with them (their methods) without taking the data out of the storage platform, in order to exploit data locality and improve performance.

In view of the above, a promising direction to take is to integrate Dislib with dataClay in order to optimize the performance of SVD. In particular, the *ds_array* can be a dataClay object composed of blocks (which are also dataClay objects). In this way, the methods associated with these objects defined in Dislib, will be transparently executed within dataClay, thus taking advantage of its features to improve performance: dataClay holds the objects already instantiated in memory, ready to receive execution requests. Then, in addition to reducing data movements, disk accesses and data transformations are also minimized, resulting in reduced execution times. This would address the requirement P1-1, and also P1-4 if a clustering algorithm, also part of Dislib, is finally considered by Pillar I.

Additionally, the matrices involved in the workflow can be transparently stored, so that they can be used in other steps of the workflow (for example from block 2 to block 3), or be reused in other executions of the workflow. This occurs without the need of explicitly transforming the matrices to a suitable encoding to be written to a file, which should in turn be read and re-constructed by the consuming step. As can be seen, this solves another source of inefficiencies, and also addresses requirements P1-2, P1-4, and P1-5.

5.2. Pillar II: Climate modelling

The ultimate goal of WP5, as reported in D5.1[20], is to exploit the eFlows4HPC architecture and software stack to enhance innovation for intelligent and integrated end-to-end HPDA-enabled ensemble Earth System Model (ESM) workflows.

Usually, an ESM ensemble experiment consists of several members divided in several sequential simulation chunks. In addition, inputs and especially outputs of the simulations runs need to be further adapted/reduced/summarized in order to:

1. Allow the execution of the subsequent analyses usually performed by different and heterogenous tools or approaches (e.g. cdo, nco, Ophidia, esmval-tool, ML procedures, etc.) depending on the nature on the operations to carry out.
2. Extract summarized information or insights, in general term knowledge, from the very large amount of data produced in order to distill relevant information for scientific evaluations or decision making purposes.

In this sense, a performant workflow solution is needed since these types of experiments are very resource-consuming in terms of computational and storage requirements; in particular, considering the latter as relevant for this document, in the following we start from the storage related requirements moving then to propose some optimizations exploiting the storage software/technologies foreseen in the context of eFlows4HPC.

5.2.1 Requirements related to storage

Among the set of requirements related to Pillar II (reported in D1.1 and D5.1), the following ones are related to storage:

Table 4: Pillar II Requirements related to Storage

P2-4	**Integration with permanent storage**	**Results may be maintained in long-term storage for archiving purposes, second use (e.g, downstream services) and/or to satisfy FAIRness policies.**
P2-6	Access to intermediate in-memory results	The workflow manager should have the capability to retrieve data/intermediate outputs of the current running members of an ensemble on execution time directly from memory.
P2-9	DA capabilities	Support for descriptive analytics (e.g., statistical analysis) exploiting fast in-memory analysis.

Concerning the requirement P2-4, the eFlows4HPC Pillar II implementation will lead to a reduction of the unnecessary model outputs, by means of a pruning of some ensemble members that will be discarded at run time; in any case, the final archiving of such outputs requires specific and large storage hardware appliances (e.g., a tape library) and usually does not affect the workflow execution time (e.g., a dedicated procedure could be run at the completion of the simulations to copy/move the selected outputs in a long term storage device). Considering this, in the context of this deliverable we will focus our attention on other aspects and the technologies selected to optimize the Pillar II workflow exploiting Hecuba and the Ophidia framework.

5.2.2 Proposed optimizations in eFlows4HPC - Ophidia

In the context of the Pillar II of the eFlows4HPC project, the Ophidia Framework will provide a new methodology for analytics and feature extraction at scale, in particular for multi-model analysis and extreme event analysis.

The following table (D5.1) contains an excerpt of the identified Pillar II macro blocks in which Ophidia will play a primary role in order to improve the performances and speed up the execution.

Table 5: Pillar II Building Blocks to be optimized

Building Block	Name	Included actions	Input/Output data structure	Description
2	Pre-processing phase	Concatenation of timesteps, regridding (if needed), variables selection, etc.	Input: CMIP6 or CMCC-CM3 datasets (NetCDF) Outputs: NetCDF files suitable for TC detection/tracking or Analytics blocks	Performs a set of preliminary steps to organize/modify/regrid the data accordingly for the following substeps. Ophidia will be used to select and concatenate the timeseries from the CMCC-CM3 outputs.
4	Multimember/Statistical Analysis	Percentile/threshold based extreme events indices computation on temperature/precipitation (e.g. heat waves, …), multi-model trend analysis, multi-model intercomparison, etc.	Input: Pre-processed CMCC-CM3 dataset (NetCDF format), statistical analysis from Feature Extraction (NetCDF format), Observational best track data Output: NetCDF, txt files, maps with indices/analytics results data	Performs a Multimember and statistical analysis operations extracting aggregated added values from the climate simulation run or from the Feature Extraction phase outputs. In addition, performs validation with respect to observations. Ophidia will be used to perform computations like: Nr of TC per basin, categorization of TC, indices extraction (es. HWDI – Heat Wave Duration Index). We have to rely on a baseline of ~30 years.

More specifically:

- Building Block nr 2 consists in the execution of various operations to adapt the outputs of the CMCC-CM3 climate model or the CMIP6 datasets and make them suitable for the subsequent Tropical Cyclone (TC) detection/tracking or for the extraction of some statistical and aggregated indices;
- Building Block nr 4 is devoted to the extraction on some extreme events indices related to the temperature or precipitation variables as well as the computation on trends or

intercomparison between datasets coming from different sources (e.g. TC detection computed by a deterministic o ML based approach).

All these operations are usually performed by means of specific climate data oriented and sequential tools (e.g. cdo, nco) or, more recently, of ad hoc python scripts exploiting the NetCDF support provided by some software libraries/packages (e.g. xarray). With respect to these, Ophidia offers a complete environment and framework specifically tailored to run in HPC infrastructures and able to natively exploit data distribution (on different nodes of the cluster) and a parallel execution (on multiple cores/threads). In these terms, parallel I/O (to speed-up the retrieval/storing of data from/to the underlying storage) and in memory parallel analysis (complex operations on multiple computational cores or concurrently on different sets of data), are key features to improve the performances of the entire workflow in the Pillar II context.

In particular, concerning the latter aspect, Ophidia provides a data distribution mechanism that allows to span the ingested datasets on different storage devices (usually RAM memory) on different cluster nodes. This allows to perform all the required data analytics operations directly in memory and in a pipelined or nested way, overcoming the overhead due to the I/O from/to the storage disk. In addition, the data distribution allows to perform such operations exploiting different processes or threads in a parallel (e.g. MPI) environment so that each task of a single HPC job operates on a predefined chunk of data. Finally, it acts as a service more than a stand-alone tool: data could be stored in memory for further manipulations/analyses when they are required, as in the case of the multimember statistical analysis where a prior computation has to be performed to extract the baseline on ~30 years of data.

5.2.3 Proposed optimizations in eFlows4HPC - Hecuba

Building block 6 of Pillar II (see deliverable D5.1) requires an analysis of the output data of the model at runtime to decide which simulations do not add value to the results and thus can be pruned.

The goal is to develop a dynamic data analysis that accesses the output data before the simulation ends execution. The combination of Hecuba with PyCOMPSs improves the productivity of the programmers by providing an easy and transparent way to exploit the underlying distributed systems. Moreover, Hecuba is implemented on top of a database with an indexing mechanism that can be used to provide fast access to the data being generated.

This use case plan is to work on two different models: OpenIFS and FESOM2. We have implemented a proof-of-concept prototype to allow OpenIFS to use Hecuba to store the data. Preliminary results showed that the simulation can benefit from the asynchronous writing interface provided by Hecuba and the underlying highly scalable distributed database. Thus, we plan to modify the implementation of both models, OpenIFS and FESOM2, to use Hecuba to store the data generated by the simulations run in building block 6 to facilitate simultaneous access from the dynamic data analysis tool.

Building block 7, consists of processing the output of the models to prepare it as input data for the data analysis tools, and building block 8 component executes the post-mortem data analysis on the output data. Traditionally these tools require the input data in NetCDF format. For this reason, we will explore the addition of an interface compatible with NetCDF into Hecuba, to evaluate whether it is possible to use it as a storage backend for this use case.

5.3. Pillar III: Urgent computing

As stated in D6.1[21], the main goal of Pillar III in eFlows4HPC is to improve and optimize workflows to provide a rapid response service to natural disasters, focusing on tsunamis and earthquakes. The aim is to deliver faster end-to-end runs, with more robust and reliable workflows, as well as more usable outcomes for potential end-users.

Pillar III is divided into two workflows: one focusing on tsunamis, called PTF/FTRT, and another one for earthquakes, called UCIS4EQ.

In the following subsections, we will first summarize the requirements that are relevant to possible storage optimizations, and then explain the proposed optimizations and how they will be addressed in eFlows4HPC.

5.3.1 Requirements related to storage

Among the set of requirements related to Pillar III (reported in D1.1 and D6.1), the following table summarizes those related to storage, shared by both workflows in the Pillar:

Table 6: Pillar III Requirements related to Storage

P3-3	Data replication	Redundancy of data is required in different phases of the workflow execution. Source data must be replicated in different locations to assure a high-availability computation as well as avoiding time consuming data transfers (e.g. computational meshes). Intermediate data generated by large computation must be also considered in order to avoid losing data in case of failures
P3-4	Execution robustness	Support for the management of fault tolerance during the workflow execution including checkpoints or retries. For example, during a large execution if a node fails, the workflow must be able to recover and continue to the end.
P3-9	Integration with permanent storage	Support for access to external data repositories (R/W) such as EUDAT Data Storage services (e.g. B2DROP). Support for final storage in long-term storage for second use and/or to satisfy FAIRness policies.

Table 3 in D6.1 summarizes the data requirements for the PTF/FTRT workflow. In this table we can observe that most of the data is manipulated in NetCDF format throughout the workflow. NetCDF is a file format widely used in earth sciences, optimized for storing and calculating time-series data. Thus, NetCDF is not only a file format, but also provides a set of functions to facilitate manipulation and perform calculations on the data stored in the files. Hence, the Python applications consuming these data do not need to map it to their own data structures in order to compute their outputs, but they can directly use NetCDF instead. NetCDF is used in this workflow for intermediate results, as well as for providing part of the output.

Similarly, Table 4 in D6.1 reports the data requirements for the UCIS4EQ workflow. There we can observe that diverse data formats, some of them structured and some of them not, are included in this workflow to represent and store the different data sets and types of data involved. Regarding the structured formats, we can see that HDF5 is considered for a majority of the

datasets. Similar to NetCDF, HDF5 is also a file format and an associated set of functions that enable the manipulation of the data, and is also widely used in scientific workflows. In this case, it is used to store intermediate results within the workflow.

5.3.2 Proposed optimizations in eFlows4HPC

Due to its capabilities to manage and manipulate NetCDF files using an in-memory approach, as mentioned in the previous section, the Ophidia Framework could also be applied in the context of the Pillar III of the eFlows4HPC project, specifically, in order to support the PTF/FTRT workflow.

More in detail, one of the needs of this use case is to manage/analyze a huge amount of data in terms of size but also in terms of file number. Considering Figure 3 in D6.1, this is particularly highlighted in blocks #3 and #4 where, starting from a large number of files (one for each tsunami scenario), some postprocessing operations are applied (#3) before a final merge of all the results (#4).

The current implementation is Python-based and it requires continuous I/O operations from disk in order to save and then retrieve the outputs for the final merging phase.

In this sense, the PTF/FTRT use case in Pillar III could benefit from the Ophidia features concerning two fundamental aspects:

1. The operations in block #3 could be performed in parallel by submitting n Ophidia jobs to the HPC scheduler preserving the outputs in memory to allow a fast retrieval for the subsequent steps. If there is the need to save these intermediate outputs, the writing on files and the subsequent operations can be carried out concurrently.

2. The operations in block #4 could retrieve the needed data directly from memory avoiding unnecessary I/O operations; in addition, the merge phase could be performed by means of a parallel Ophidia operation running on multiple cores or threads and addressing many bunches of data at a time.

An alternative storage optimization for this workflow would be to transparently replace files by dataClay objects, in order to exploit the in-storage computation capabilities of dataClay. The idea would be to provide those objects with a NetCDF interface, and implement NetCDF functions on them, so that they can be executed without the need to transfer the data to the application space. This would be applied to those steps that are executed within an HPC infrastructure, which according to the current definition of the workflow, correspond to blocks #3 to #5. However, this optimization will only provide some advantage if the envisioned time spent in data access is very high with respect to the time spent in calculations. As some of the blocks are not yet implemented, Pillar III is currently estimating this ratio so that a decision on this possible optimization can be made.

Regarding the UCIS4EQ workflow, no clear possibilities of optimization have been identified at the moment. In UCIS4EQ data is shared between blocks across different infrastructures. In this scenario, data transfers cannot be avoided, and in-memory approaches cannot be exploited to improve performance, which are precisely the main benefits of using the alternative storage solutions considered in eFlows4HPC.

6. Conclusion

Results from the previous sections will contribute to the design and optimisation of the workflow management stack and of the workflows themselves of the Eflow4HPC project.

Regarding the task of **optimizing the usage of Data Analytics and Machine Learning libraries, and of the workflow runtimes and orchestration engines**, the area of investigation seems huge: Data preparation, data transformation, data movement, data partitioning, algorithms (hyper) parameters tuning, orchestration (optimizing resource usage and tasks parallelism).

Therefore, it has been decided to focus for now on two specific problems, data partitioning and workflow orchestration.

For the first problem, data partitioning (or chunking), there are two subproblems: data chunk placement and data chunk size. The second one has been taken as an example for an optimisation strategy definition. The strategy will be to design a Machine Learning model that will be able to estimate the optimal chunk size, and to apply it first with the Dislib distributed data processing framework that will be used in the project. Later the results can be transferred to other libraries using chunking.

For the second problem, workflow orchestration, the strategy will be to evaluate and provide some means

- to optimize the parallel execution of workflow execution steps, e.g., by detecting dependencies between different executions as well as by coordinating the use of resources done by the different runtimes involved in each execution,
- and to study how a more dynamic resource management could enhance a parallel workload execution, e.g., by predicting an application workload.

This will be studied first focusing on the COMPSs capabilities to achieve such goals.

Regarding the **containerization** strategy, the approach has been first to list all eFlows4HPC software components and check their availability as part of a containerized distribution; then to study how the components in charge of workflows deployment are using container technologies; and finally to take into account all constraints related to the usage of container technologies.

The first phase shows that although most of the components are available through container-based distribution, special attention should be paid to some of them, either building ad hoc container images, or adopting a particular deployment strategy to cope with some specific constraints.

The second phase has described how both components in charge of workflow deployment, Yorc at the higher level and PyCOMPSs at the lower level, can build upon container technologies to manage workflows deployment. Yorc will be used to both deploy jobs as Singularity Containers on HPC systems, and any Docker based application on Cloud. PyCOMPSs applications can be deployed as containers and PYCOMPSs itself may execute applicative tasks in containers: the strategy will combine both approaches while proposing automation tools for image building and optimizing the execution pattern. In addition, it is proposed that FastML, which is a tool for AI model training management, will be used for ML steps of workflow deployment, as it facilitates the deployment of ML jobs on HPC clusters.

Finally, constraints due to container technologies usage that may affect the project have been highlighted, including restriction of usage on a particular technology (Docker or Singularity), and

potential performance bottlenecks related to target architecture compilation, network, etc. For this last point, the strategy will be to pay attention to the image building process in order to provide images which are suited for the target infrastructure architecture, either at the container engine level, or at deployment time; corresponding methodology and/or automated tools will be studied and provided for this purpose.

Regarding **Storage technologies**, the strategy is to study alternative storage technologies in order to improve workflow execution performance. This is done in the context of each Pillar workflow, according to the related requirements.

For Pillar I, the direction that will be taken is the integration of the Distributed Data Store Dataclay with the distributed Computing Library Dislib, which will reduce data movement, disk access, and data transformation.

For Pillar II, it is proposed:

- on one side to use Ophidia in order to optimize the execution workflow parts related to data ingestion, transformation and analysis, thanks to its distributed data storage capabilities,
- and on the other side to use the Hecuba storage system to enable dynamic data analysis rendering easy and efficient access to the output data of a model at runtime.

For Pillar III, two approaches are proposed, using Ophidia to parallelize operations on data to be handled in memory, and using Dataclay to benefit from in-storage computation capabilities.

The strategies elaborated during these studies and described in this document will highly contribute to the workflow improvements targeted in the project and will more specifically have an impact on the following metrics, extracted from the D1.1[1] document:

Table 7: Metrics impacted by D2.2 Strategies

Acronym	Name	Description	D2.2 contribution
DoP	Degree of Portability	Percentage of workflow components that can be reused in other infrastructures and workflows.	Container strategy
DT	Deployment Time	Time elapsed to deploy the workflow.	Container strategy
ET	Execution Time	Time elapsed to execute a workflow.	DA/ML/Workflow optimisation, Storage technologies
SU	Speed-up	Execution time improvement when running with larger resources. Calculated as: SU(N) = ET(base)/ET(N) where ET(base) is the baseline and ET(N) is the execution with N times larger resources.	DA/ML/Workflow optimisation, Storage technologies

Eff	Efficiency	Execution time degradation when running larger problems. Calculated as: $Eff(N) = ET_{base}(base) \ / ET_N(N)$ where $ET_{base}(base)$ is the execution of the baseline problem and infrastructure and $ET_N(N)$ is the execution time of N times larger problem and infrastructure.	DA/ML/Workflow optimisation, Storage technologies
TD	Transferred Data	Amount of data transferred (in bytes) by the workflow between different compute nodes of the computing infrastructure.	Storage technologies
DM	Data Movements	Number of transfer operations between different compute nodes of the computing infrastructure.	Storage technologies
IOT	I/O Time	Percentage of Execution time performing I/O operations.	Storage technologies
CH	Core/Hour	Number hours of a CPU Core consumed by the workflow execution.	DA/ML/Workflow optimisation
EC	Energy Consumption	Energy consumed (Wh or Joules) associated with a workflow execution.	DA/ML/Workflow optimisation

Of course, all these workflow execution performance improvements will also have an impact on the Pillar specific metrics which are defined in D4.1[19], D5.1[20], D6.1[21].

For Pillar I, it will mainly contribute to ROMTime (Time to compute a Reduced Order Model), ROMSize (size of the problem that can be reduced), SVDS (Speedup of SVD Extraction) and SVDL (Largest possible SVD) metrics.

For Pillar II, it will contribute to AR (Accuracy of the results) and SYPD (Simulated years per Day) metrics.

For Pillar III, it will mainly contribute to RT (Response Time) metric.

7. Acronyms and Abbreviations

- ABI - Application Binary Interface
- AI - Artificial Intelligence
- API – Application Programming Interface
- CA – Consortium Agreement
- CLI- Command Line Interface
- D – deliverable
- DA - Data Analytics
- DL – Deep Learning
- DoA – Description of Action (Annex 1 of the Grant Agreement)
- EB – Executive Board
- EC – European Commission
- GA – General Assembly / Grant Agreement
- GUI - Graphical User Interface
- HPC – High Performance Computing
- HPCWaaS - HPC Workflow as a Service
- IPR – Intellectual Property Right
- ISA - Instruction Set Architecture
- KPI – Key Performance Indicator
- M – Month
- ML – Machine Learning
- MS – Milestones
- OS – Operating System
- PM – Person month / Project manager
- RDMA - Remote Direct Memory Access
- TCP - Transport Control Protocol
- WP – Work Package
- WPL – Work Package Leader

8. List of Tables and Figures

9. References

[1] D1.1 Requirements, Metrics and Architecture Design, eFlowsR4HPC Deliverable, 30/06/2021

[2] Venish, A., and K. Siva Sankar. "Study of chunking algorithm in data deduplication." Proceedings of the International Conference on Soft Computing Systems. Springer, New Delhi, 2016.

[3] Cantini, Riccardo, et al. "Exploiting Machine Learning For Improving In-Memory Execution of Data-Intensive Workflows on Parallel Machines." Future Internet 13.5 (2021): 121.

[4] http://hadoop.apache.org/

[5] Tejedor E, Becerra Y, Alomar G, et al. PyCOMPSs: Parallel computational workflows in Python. The International Journal of High Performance Computing Applications. 2017;31(1):66-82.

[6] J. Álvarez Cid-Fuentes, S. Solà, P. Álvarez, A. Castro-Ginard and R. M. Badia, "dislib: Large Scale High Performance Machine Learning in Python," 2019 15th International Conference on eScience (eScience), 2019, pp. 96-105.

[7] https://hub.docker.com/r/compss/compss

[8] Ramon-Cortes, C., Serven, A., Ejarque, J., Lezzi, D., & Badia, R. M. (2018). Transparent Orchestration of Task-based Parallel Applications in Containers Platforms. Journal of Grid Computing, 1(16), 137-160. https://doi.org/10.1007/s10723-017-9425-z

[9] MPI executions in Singularity. https://sylabs.io/guides/3.5/user-guide/mpi.html

[10] Accessing NVIDIA devices from Docker. https://github.com/NVIDIA/nvidia-docker

[11] Accessing GPUS from Singularity. https://sylabs.io/guides/3.5/user-guide/gpu.html

[12] Building container images with EasyBuild. https://docs.easybuild.io/en/latest/Containers.html

[13] European Environment for Scientific Software Installations. https://eessi.github.io/docs/software_layer/

[14] Archspec Python Library. https://archspec.readthedocs.io/en/latest/

[15] Buildah. https://buildah.io/

[16] Docker Buildx. https://docs.docker.com/buildx/working-with-buildx/

[17] J. Martí, A. Queralt, D. Gasull, A. Barceló, J. J. Costa, T. Cortes: dataClay: a distributed data store for effective inter-player data sharing. Journal of Systems and Software 131, 129-145 (2017), doi: https://doi.org/10.1016/j.jss.2017.05.080

[18] D. Elia, S. Fiore and G. Aloisio, "Towards HPC and Big Data Analytics Convergence: Design and Experimental Evaluation of a HPDA Framework for eScience at Scale," in IEEE Access, vol. 9, pp. 73307-73326, 2021, doi: 10.1109/ACCESS.2021.3079139.

[19] D5 Requirements on the eFlows4HPC software stack from Pillar I and evaluation metrics, 30/06/2021

[20] D5.1 Requirements on the eFlows4HPC software stack from Pillar II and evaluation metrics, 30/06/2021

[21] D6.1 Requirements on the eFlows4HPC software stack from Pillar III and evaluation metrics, 30/06/2021