D4.1 – System Requirements Analysis for Orchestrator Design

Deliverable ID	D4.1
Deliverable Title	System Requirements Analysis for Orchestrator Design
Work Package	WP4
Dissemination Level	PUBLIC
Version	0.7
Date	2022 – 02 – 28
Status	Submitted
Deliverable Leader	LINKS
Main Contributors	LINKS

The ACROSS project has received funding from the European High-Performance Computing Joint Undertaking (JU) under grant agreement No 955648. The JU receives support from the European Union's Horizon 2020 research and innovation programme and Italy, France, Czech Republic, United Kingdom, Greece, Netherlands, Germany, Norway.

Published by the ACROSS Consortium

Document History

Version	Date	Author(s)	Description
0.1	2021-01-15	LINKS	ToC (Draft)
0.2	2021-02-06	LINKS	Content to the main sections was added
0.3	2021-02-22	LINKS	Comments and feedbacks from P. Viviani and G. Vitali (LINKS)
0.4	2021-02-23	LINKS	Updated version of the content
0.5	2021-02-28	IT4I, ATOS, SINTEF	Document review: feedbacks and comments have been provided
0.6	2021-02-28	LINKS	Reviewers comments/feedbacks addressed
0.7	2021-02-28	LINKS	Submitted

Table of Contents

Glossary

Acronym	Explanation
AI	Artificial Intelligence
ANN	Artificial Neural Network
CPU	Central Processing Unit
CSV	Comma Separated Value
CWL	Common Workflow Language
DAG	Directed Acyclic Graphs
DAM	Dynamic Allocation Module
DL	Deep Learning
ERT	Ensemble-based Reservoir Tool
FPGA	Field Programmable Gate Array
FMLE	Fast Machine Learning Engine
GPU	Graphic Processing Unit
HPC	High-Performance Computing
HPDA	High-Performance Data Analytics
ICON	Icosahedral Nonhydrostatic weather and climate model
IFS	Integrated Forecasting System
HW	Hardware
LES	Large Eddy Simulation
ML	Machine Learning
NNP	Neural Network Processor
SNN	Spiking Neural Network
SW	Software
TPU	Tensor Processing Unit
URANS	Unsteady Reynolds Averaged Navier Stokes
WRF	Weather Research and Forecasting model

List of figures

List of tables

Executive Summary

The ACROSS project targets to effectively support the execution of workflows composed of a mix of computations that spans across different computing stacks; specifically, machine learning (ML) and deep learning (DL), high-performance big data analytics (HPDA), and numerical simulations on high-performance computing (HPC) systems. To this end, one of the key elements of the ACROSS platform is found in the design and implementation of a cross-stack, multi-layered workflow orchestration system. Within the ACROSS project, this orchestration system has the role of properly managing the distribution of the workflow operations (i.e., the workload) among the available infrastructural resources. Aiming to address the energy efficiency challenge, the ACROSS platform strongly relies on a large set of heterogenous resources, spanning from different CPU architectures to various types of accelerators (including GPUs, FPGAs, Neural Network Processors – NNPs, etc.) to different types of memory devices. As a matter of such diversity, the orchestrator must apply appropriate strategies to make efficient use of such accelerators.

One of the key aspects involved in the design of the ACROSS orchestrator is given by the analysis of the *system requirements*, which allows to identify the most relevant features that the orchestrator design should provide. System requirements include requirements that touch different aspects involved in the execution of a workflow. These can be categorized as platform integration requirements, workflow specific requirements, pilot specific requirements, and performance and energy-efficiency requirements. The main purpose of this deliverable, thus, is to provide such analysis. As the main result of the analysis, the high-level architecture specification of the ACROSS orchestrator is also given (this is an update with regards to the initial high-level version provided in the D2.2), which includes a clear path towards the integration of the ACROSS orchestrator with other execution platforms.

Position of the deliverable in the whole project context

This deliverable is the first concerning the activities carried out in the context of WP4. Specifically, this deliverable covers an initial phase of collecting and analyzing system requirements that drive the choices concerning the design and implementation of the ACROSS orchestration system and its integration within the available infrastructures. As such, this deliverable is expected to allow deriving the main (high-level) architecture of the ACROSS orchestrator. Collecting system requirements demands for a close interaction with other work packages. Interaction with pilots (WP5, WP6 and WP7) allowed to collect those requirements that are specific of the execution of pilot workflows; for instance, all the software components that the orchestrator must launch, the set of (heterogeneous) resources required, etc. WP2 provided a general picture of the ACROSS execution platform, more specifically in terms of HW/SW technologies at disposal and pilot workflows in-depth description. WP3 provided useful feedback on the way HW accelerators will be used by pilot workflows. This latter is important to define a proper way of distributing the workload on different resources depending on the exposed features.

Description of the deliverable

An introductory section is provided to position both the content of this deliverable, and the whole ACROSS orchestrator in the main context of the ACROSS project. Then, the deliverable has the main purpose of collecting all the system requirements which will drive the design and implementation choices for the ACROSS orchestrator. As such, the deliverable starts with an in-depth analysis of these requirements. Given the large diversity of aspects caught by system requirements, these latter are grouped in four main categories, which are singularly analyzed. The analysis of the requirements allows the WP4 to make design and implementation choices concerning the architecture of the ACROSS orchestrator. The outcomes of these choices are represented by the (updated with regards to the initial version described in D2.2) high-level architecture described in-depth in the third section of this document. The last section summarizes the main idea brought by this document and sketches the next concrete actions towards the ACROSS orchestrator implementation.

1 Introduction

The last decade saw the emerge of a large variety of computing architectures with the aim of overcoming performance and energy inefficiencies of conventional CPUs when specific workloads are executed. An example can be found in what is known as *the Cambrian explosion in the number of specialized architectures* for accelerating machine learning (ML) and more specifically deep learning (DL) models. Pioneers of specialization are the GPUs, which have been looked at as an efficient way to provide horsepower for math operations found in numerical simulations and later for accelerating ML/DL applications. Following the path tracked by GPU vendors, other architectures emerged in the latest years, which exploit even more the specificities of ML/DL applications to further gain performance and energy efficiency (e.g., many vendors offer Neural Network Processors –NNPs, Google designed its own Tensor Processing Unit –TPU). Field Programmable Gate Arrays (FPGAs) offer the advantage of being adaptable to different application context, thanks to their unique capability of reconfiguring the fabric resources to match with the application demands.

Last but not least, the supercomputing industry has recognized the undoubted advantage of using specialized architectures for different workloads in order to gain performance and energy efficiency, and ultimately to exceed the Exaflops barrier. Also, different access models with regards to the traditional batch access are gaining interest: one above all, the Cloud model. Here, resources are provisioned on-demand (using virtual machine and container technologies), and the management system tries to optimize their usage for a multitenancy context. As a matter of fact, a growing number of applications that leverage supercomputers started to largely exploit hardware acceleration, and the use of Cloud resources is considered useful in supporting specific steps of the workflow execution (e.g., visualization, data pre-processing, etc., which are found not demanding high parallelism in terms of number of cores). Furthermore, the integration of specific execution steps that make use of ML/DL techniques or high-performance big-data analytics (HPDA) in many of these applications has been found providing great benefits. We can refer to these workflows as *cross-domain workflows*: their efficient execution on current generation and upcoming supercomputers poses big challenges to the whole software stack that oversees abstracting the underlying infrastructure, provisioning the computing, storage, and networking resources, and allocating jobs on such resources. Indeed, batch schedulers are generally used to serve as the main access point to supercomputing resources; however, despite their sophistication, priority functions used to schedule jobs' execution poorly adapt to very heterogeneous environments. They are also difficult to adapt over the time to the underlying infrastructural changes.

The ACROSS project aims to address such challenges, and to this end, the design and implementation of a cross-domain, multi-level orchestration system is of paramount importance. As such, the ACROSS orchestrator will be able to take advantage of the variety of computing architectures made available by IT4I and CINECA supercomputers by moving the software managing resource provisioning and jobs allocation from a monolithic design to a multilevel modular one. The computing resource assignment phase, the deployment of specific components required for the execution of the workflows and the actual workflow execution are controlled by specific software components. The devised architecture of the ACROSS orchestrator will allow for a finer-grain management of the computing resources (e.g., assigning a subset of the CPU cores of a node to a given job), thus allowing for a better exploitation of resources. Describing these complex cross-domain workflows is done through a language that can catch all their specificities (high expressiveness). Finally, the ACROSS orchestrator will also exploit the Cloud computing access model to better match with the pilot workflows' requirements.

1.1 Scope

The scope of this document is to collect all the *system requirements* that are expected to drive the design and implementation choices for the ACROSS orchestrator and provide a detailed analysis of them. Given the complexity of creating such an orchestrator and the complexity of the workflows provided by Pilots' application use cases, system requirements can be made falling into various categories: platform integration, workflow specific, pilot application specific, and performance and energy efficiency. Concerning the platform integration category, here, we consider those requirements that are connected to the integration of the orchestrator with the infrastructural level. Here, we also consider how the orchestrator can be integrated with other execution environments. Workflow specific requirements are related to the way workflows must be described and submitted to the ACROSS platform. Pilot specific requirements covers all those requirements that are strictly connected to a specific pilot workflow (e.g., installation and interaction with specific software). Finally, the performance and energy efficiency category collects the requirement related to the management of heterogeneous computing and storage resources, which are at the basis of achieving the expected performance and energy consumption reduction.

The analysis of such requirements converged into a first version of the orchestration system architecture. As such, the purpose of this document is also that of providing the description of the ACROSS orchestrator by highlighting how the various software modules identified in the deliverable D2.2 are connected and how this interconnection allows to fulfil the collected requirements. Also, it provides a clear pathway for the integration of the ACROSS orchestrator with external platforms that may allow to cover aspects that are not the primary focus of the project (e.g., secure access to the resources, federation, etc.). Through this it will be possible to further extend the range of potential applications and infrastructural resources that will be served by the ACROSS platform.

1.2 Related documents

ID	Title	Reference	Version	Date
[RD.1]	Summary of Pilots co-design requirements	D2.1	3.0	31-08-2021
[RD.2]	Description of key technologies and platform design	D.2.2	0.8	30-11-2021

2 System Requirements

The ACROSS project aims at supporting the execution of *cross-stack multi-domain workflows*, as modern applications are expected to mix at different steps of the execution numerical simulations, deep learning and machine learning tasks, big-data analytics. Exascale machines will be more complex than past generations of supercomputers, by putting together different types of accelerators, I/O systems, and storage devices. Diversity is also expressed by the availability of different provisioning models for the resources: on one side, getting access to HPC resource is mediated by batch schedulers; on the other hand, the Cloud model can be used to provide access in a more flexible way (resources are virtualized, and they are instantiated and relinquished on-demand).

Such diversity is at the basis of improving energy efficiency and performance; however, it requires that the Resource and Job Management System (RJMS), we also refer to this as the **orchestrator**, evolves accordingly. To this end, in the ACROSS project, the orchestrator has been envisioned as a multi-level system:

- The *high-level layer* which is responsible for the definition of the workflows.
- The *middle layer* which is responsible for the acquisition of the resources (both Cloud and HPC) and the execution of workflows by properly scheduling the execution of jobs.
- The *low-level layer* which is responsible for the optimal execution of jobs by exploiting hardware acceleration, low-level scheduling strategies, etc.

While the ACROSS orchestrator's objectives are focused on performance and energy efficiency, other factors should be accounted for during all its design and development phases. Some of them can be found in the secure access to the computing/storage resources and in federating computing and storage resources (here, workflows or specific portions, can be executed on resources located in data centers that may be geographically separated). Given that all, the ACROSS orchestrator design puts strong emphasis on **modularity**: there is an orchestrator core which is the main target of the work carried out in the WP4 and it exposes the right interface(s) to allow the connection with external platforms, which is expected to be explored by Task T2.5.

Bringing together all these concepts, we organized the ACROSS orchestrator architecture in different layers and, as an example of integration with external execution platforms, we also sketched a possible way of integration with the LEXIS federated platform [1].

In the following, we analyze all the requirements that are at behind the design concepts for the ACROSS orchestrator. Such requirements cover different aspects, ranging from those related to the overall platform and the infrastructural resources made available by CINECA and IT4I, to those specific of the pilots, passing through the requirements related to the workflow description and heterogeneity awareness.

2.1 Platform integration requirements

The ACROSS platform is identified in the full software stack that is responsible for driving the execution of the workflows and in the hardware components that provide the execution substrate and the horsepower for performance boost. As the user must execute jobs on the supercomputer resources, he has to be granted to access the login node of the system and submit the jobs to a batch scheduler. This latter is in charge of acquiring the needed resources and planning a specific point in time when to execute the job. The management of the resources, as well as the submission to the batch scheduler are mediated by a set of queues which allow the administrator of the supercomputers to apply their scheduling policies and to differentiate on the access priorities of the resources. While this system provides a simple and effective interface to the computing resources, it is not sufficiently flexible to effectively sustain the execution of cross-domain workflows. For instance, the granularity at which resources can be managed is that of the computing node, which may also contain accelerators. As such, in most of the cases where the workload is unbalanced (e.g., there is a large portion of the job executing on the accelerator and very small one using the CPUs) or only a fraction of the cores is needed, this quickly leads to a suboptimal use of the resources. Another important aspect of dealing with a queue-based scheduler concerns the waiting time of enqueued jobs, that is generally not negligible in comparison to their maximum wall clock time. Also, jobs exceeding the maximum execution time accountable for jobs in a certain queue, must be checkpointed and restarted, thus incurring in multiple waiting periods in the same submission queue (see Figure 1 –the estimated duration of the submitted job (left) is split in two separated submissions (right), each having its own queue waiting time Q_T). Indeed, for these jobs, it is expected that all new resubmissions of a checkpointed job will incur in a not negligible waiting time in the queue.

All these factors may greatly influence the execution of the workflows and limit the amount of work that can be executed concurrently. Better allocation of the resources over the time (e.g., exploiting the advance reservation mechanisms of the batch schedulers) may greatly reduce such limitations.

Also, to overcome such limitations, many works in the literature proposed approaches to enhance the conventional batch schedulers. Some of them have been of inspiration for the ACROSS orchestration architecture. In [2], the authors describe a hierarchical workflow management system (Flux) which has been developed to maximize the utilization of heterogeneous computing nodes by being able to co-locate jobs using GPUs and jobs relying only on the CPUs on the same set of nodes. Also, Flux provides an effective mechanism to allocate resources and allow scheduling agents running on them to gather the work to perform from a shared in-memory database. A similar concept has been explored in [3] with the Balsam orchestration system. The authors of [4] combine different software technologies including Balsam (for managing multiple HPC execution location), Flux (for managing workflows on each location) and Parsl to ease the workflow automation.

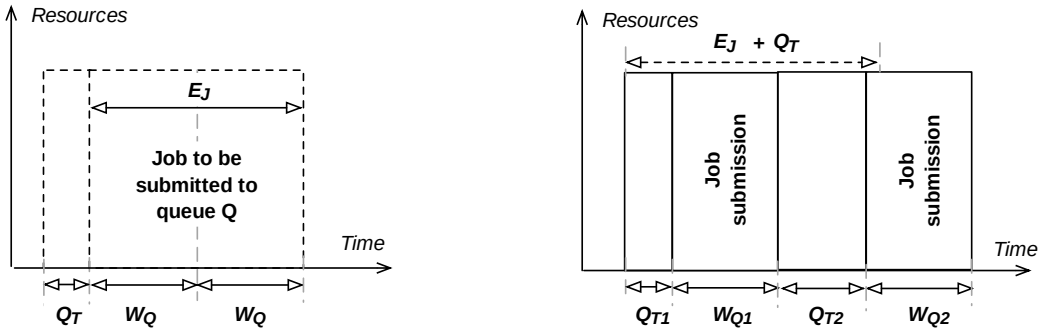

Figure 1 - (Left) Estimated duration (E_J) of the job to be submitted on the generic queue Q and the maximum wall clock time allowed for jobs in queue Q. (Right) the actual execution of the job which is split in two separated submissions (W_Q), each with an additional queuing time (Q_T).

Like these approaches, the ACROSS orchestrator has been designed around different components that ease the workflow description, execution, and the exploitation of heterogeneous computing resources. To accomplish this, the ACROSS orchestrator has been thought around the capability of the underlying batch scheduler to expose a (advanced) resource reservation mechanism. The internal logic of the ACROSS orchestrator can plan the reservation of the appropriate number of resources for maximizing their usage and minimizing the workflow completion times. Another important aspect to consider in designing our solution concerns the ownership of the execution of different components of the orchestration stack. Indeed, only few of them are persistent (i.e., are not bounded to the single user session –see Section 3), while the others are either launched at the time of logging on the cluster frontend node (i.e., login node) or by other components already running. As such, it is important that all these components have the correct permissions set up.

Dealing with workflow steps that use DL models brings additional requirements. As such, ANNs are characterized by a set of parameters (hyperparameters) that must be properly tuned in order to allow the model to correctly work. This initial phase is generally managed through an iterative process that often requires the user intervention (e.g., to monitoring the convergence of the training, chancing some of the tuneable parameters). To properly catch this situation, the ACROSS orchestrator has been devised in such a way these interactive sessions could be possible. To this end, the Fast Machine Learning Engine (FMLE) component described in Section 3 provides all the necessary capabilities to fulfil with this requirement.

2.2 Workflow specification requirements

By layering the orchestrator architecture, we aim at keeping away the complexity of managing the underlying computing, storage, and networking resources from the user, as well as the execution of the various steps composing the workflow. As such, we expect to provide the users with a simple and flexible interface to define their workflows. One of the major problems that one encounters in this abstraction process is that of limiting the expressiveness left to the users for describing their workflows. Indeed, it is required to define the workflows as:

- The sequence of steps to be executed, along with their mutual dependencies.
- The environment (i.e., the specific software stack and the specific resources –e.g., the number of computing nodes, the number of GPUs, etc.) that is required by each step to be correctly executed.

Considering the definition of the workflow steps, a set of important features becomes of value for supporting the users. Workflows (i.e., steps sequence) should be provided in a declarative manner, and the user should

be able not only to define dependencies among the steps, but also express concurrency and parallelism. Other interesting features concern the conditional execution (i.e., a certain step is executed upon a specified condition happens) and the capability of managing data streams (e.g., a certain step executes whenever new data on the input streams becomes available); as well as a convenient way of describing and associating the specific execution environment to each workflow step should be provided. Last but not least, a simple mechanism to submit and monitoring the execution of the workflows should be provided.

All these features guarantee the high level of expressiveness required to properly abstract the underlying orchestration mechanics, without negatively impacting the user experience. The ACROSS orchestrator will expose such features through the high-level layer. The frontend object of the Workflow-aware Advanced Resource Planner (WARP –see Section 3) provides a simple command line interface abstracting the underlying mechanism for reserving in advance computing resources and executing submitted workflows. On the other hand, an approach [5] based on the Common Workflow Language (CWL) allows the users to easily define their workflows and to match these with the required resources.

2.3 Pilot-specific requirements

The pilot applications have their specific software tools that are required to run the workflows. Specifically, these tools are used to drive the execution of the workflow, manage ensemble simulations, and to perform specific computations (e.g., Damaris –i.e., a middleware for I/O and data management targeting large-scale MPI-based HPC simulations, ERT, Kronos scheduler, etc.). Furthermore, there are also specific demands concerning the usage of Cloud computing resources that may be used in some of the workflow steps, as well as the support for managing iterative phases of the workflow execution.

In the following subsections, we analyse these specific pilot-focused requirements.

2.3.1 WP5 – Aeronautics

The Aeronautics pilot concerns the execution of application workflows linked to two use cases: one use case concerns the design and optimization of an aeronautic turbine; the second use case is linked to the design and optimization of the engine combustor (see deliverable D2.1). Both these use cases imply to take into consideration some specific requirements, as described in the following. The ‘turbine’ workflow relies on two different simulations (LES and URANS) to fulfill the design and optimization process, as well as on HPDA processing steps. In particular, this use case workflow sees the execution of the LES simulation which periodically generates input files (in the form of CSVs) that must be processed by the HPDA procedure. This is a processing step that does not require large number of nodes and performs some linear algebra operation on the input values. Here, the major requirement (similar to the following WP6 case) is being able to schedule the execution of two jobs (i.e., the LES and the HPDA) in such a way they will overlap in time. We refer to this scheduling schema as *job co-scheduling*. To accomplish with this requirement, the WARP module will be of help since the internal logic will try to create proper resource reservations. Deliverable D2.1 described in detail this use case; one of the important remarks is that the ANN models being part of this workflow need to be trained using data coming from the URANS simulations with an iterative process. This implies a major difficult in defining the workflow, since generally workflow structure should resemble that of a Directed Acyclic Graph (DAG). As such the workflow structure and the orchestrator should take care of the iterations needed to train the ANNs and to assess their convergence.

Looking at the other use case (also well described in D2.1), the workflow appears less complex than the turbine case, since there are simulation jobs covering different physical domains that run in parallel. However, the application relies on a proprietary tool that simplifies the management of the simulation execution. Beside this, some pre- and post-processing phases could fit on Cloud resources, and thus the orchestrator would be demanded to properly manage their execution on such type of resources.

2.3.2 WP6 – Weather and Climate

The Weather and Climate pilot concerns the execution of application workflows that are at the basis of weather forecasts and climate simulations. Within WP6, which foresees developing and testing such workflows, some important requirements have been identified for the ACROSS orchestration design.

Specifically, there are two workflows that are challenging from the orchestration viewpoint:

1. Climate simulation with the ICON model, where two separated components should be optimally placed on the machine: i) the atmosphere simulation which is a completely GPU-based software; and ii) the ocean simulation which is a completely CPU-based software. Both the simulations are very expensive and requires a very powerful machine (i.e., the number of nodes required is high). Here, the optimality criteria is given by the fact that, in the best case the two simulations can be located on

the same group of resources (each using a subset of the resources on each node) and running in parallel.

2. Down-streaming workflow, where there is a software component (the IFS weather model) which is responsible for running a simulation and generating data (stream) which is supposed to be consumed by another software component (product generator) once the subset of the data stream is generated.

Figure 2 shows the two situations arising from these two challenging workflows, which demand, on one side, the co-scheduling of separate jobs, and the co-location of jobs, on the other ones. More specifically, the challenge arising from the first case lies in the optimal allocation (i.e., possibly, placing these two simulations on the same group of HPC nodes) of the two software (atmosphere and ocean). Indeed, theoretically they may use completely diverse sets of resources (on one side CPUs and on the other GPUs plus few CPU cores), which suggests for the exploitation of an (optimal) co-location of the two simulations. The idea that could be explored, would be of making the heterogeneous nodes running ocean parts on the CPU part of the nodes (it is a large MPI application) and the atmosphere greatly exploiting the GPU acceleration. The co-location, however, comes with some big challenges to be solved. First, the ocean application as well as the atmosphere one, both use an I/O server running on the nodes to manage I/O operations. The traffic that is generated by the application during the computation may affect the I/O traffic, that translates in poor performance of the application. The problem could be solved partially, by moving the I/O server on a separated group of nodes, thus paying only for the time of transferring data to the server but avoiding further interference with simulation data exchange. However, the data movement between GPU and CPU part still may affect the networking performance and thus may impact on the overall application performance. A deep understanding of these mechanisms is required to improve the orchestration and allow for the exploration of co-location.

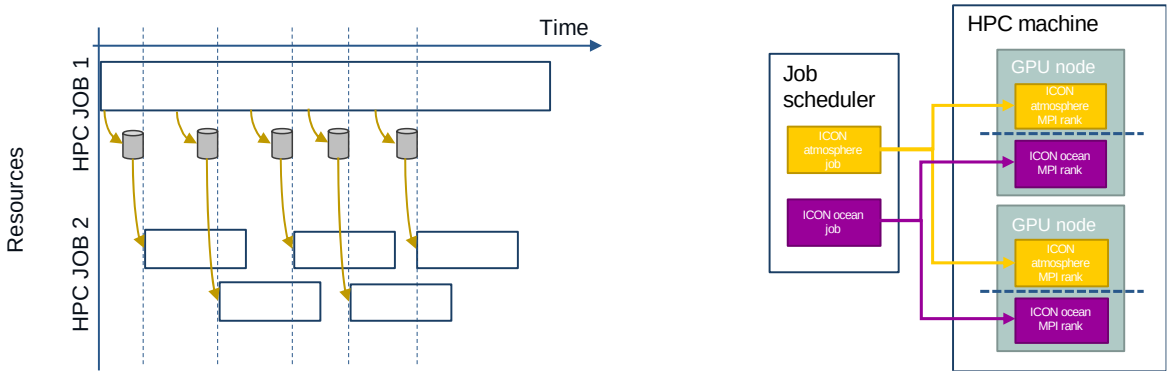

Figure 2 - (Left) separated HPC jobs must be co-scheduled (their execution should be overlapped in time); (Right) separated jobs must need to be co-located on the same set of resources.

For the second case, the challenge is in the capability of the orchestrator to trigger the execution of downstream jobs (e.g., WRF [6]). This is quite different from the way workflows generally are managed, since the progress from one step of the workflow to the next is triggered by control signals (the jobs complete, and the batch scheduler can expose such information to the orchestrator –the signal). StreamFlow [5] has been selected as one of the key components of the ACROSS orchestrator since it provides high flexibility with regards to the management of workflows. As such, the workflow description will be enhanced to allow such data-triggering.

Other aspects to be considered concern the creation of a deployment plan for the specific components that will be used in these workflows (ICON, IFS, Kronos scheduler, etc.), and their interface with the ACROSS orchestration core components.

2.3.3 WP7 – Energy and Carbon Sequestration

This work package entails the execution of workflows linked mainly to ensemble simulations. The management of these ensembles is done through a pilot-specific tool (ERT). Also, within this workflow, the ACROSS partners would like to explore the capability of performing in-situ and in-transit processing; this is achieved by means of another pilot-specific tool, named DAMARIS [7]. The main requirements concerning this work package concern the proper integration of the ERT [8] and DAMARIS tools within the ACROSS orchestrator architecture. Further activities will be foreseen in the context of managing the execution of some parts of the workflows eventually in Cloud.

2.4 Performance and energy efficiency requirements

One of the ways to achieve greater energy efficiency is the exploitation of HW acceleration. Beside conventional GPUs, the ACROSS project aims at exploring much more advanced technologies. As such,

neuromorphic chips have been identified as one direction where to investigate. Neuromorphic chips along with Spiking Neural Networks (SNNs) show very promising capabilities over than a greater energy efficiency compared to traditional Artificial Neural Networks (ANNs), that are complex to reproduce with conventional ANN models. As a matter of that, the ACROSS orchestrator should be aware of such type of devices and their accompanying SW toolchains. To address this point, the orchestrator architecture has been extended to integrate a specific component (FMLE –see Section 3) that facilitate modelling, training, and execution of both ANN models and their SNN counterparts.

The process of developing and testing the execution of a workflow implies the need for a mechanism for understanding where (eventually) the execution failed. While a 'debugging' system is out of the scope of the orchestration system, a way for logging the activity of different components becomes mandatory. As such, all the orchestrator's components should provide such a mechanism. Through the logged information, the user has the capability of reconstructing the execution flow and can determine the possible sources of failure.

2.5 Summary of the system requirements

For the sake of clarity, all the requirements discussed in the previous subsections are summarised in the Table 1.

Table 1 - Summary of the system requirements

Requirement ID	Category	Description
R1	Platform integration	Exploitation of the great hardware heterogeneity available with available HPC machines (GPUs, FPGAs, NNPs, etc.)
R2	Platform integration	Capability of managing HPC computing resources with a finer granularity than that of a node (i.e., allocation of subset of CPU cores or a subset of accelerators within a node)
R3	Platform integration	Capability of seamlessly using HPC and Cloud resources in the same workflow; on the Cloud side, the capability of automatize the process of deploying/undeploying the execution environment required by the workflow's steps
R4	Platform integration	Exploiting advanced resource reservation features of the batch schedulers (SLURM, PBSpro)
R5	Platform integration	Modularity of the orchestrator in order to be integrated also with other external platforms (e.g., LEXIS).
R6	Platform integration	Ease the management of resources involved in the modelling and training phases of a ML/DL model
R7	Workflow specification	Being able to describe the workflow steps with a high expressiveness (i.e., capability to define concurrent steps, parallel steps, conditional steps, etc.)
R8	Workflow specification	Being able to describe and bind the execution environment for each step of a given workflow
R9	Pilot-specific	Being able to co-scheduling and co-allocating jobs
R10	Pilot-specific	Integration of pilot-specific low-level tools
R11	Performance and energy efficiency	Improving application performance and overall energy efficiency through the exploitation of heterogeneity available with available HPC machines, with a focus on neuromorphic devices
R12	Performance and energy efficiency	Integration of the neuromorphic toolchain with the orchestrator core components

3 ACROSS Orchestrator Architecture – Updated vision

The fulfilment of all the previously stated requirements requires to properly connect different software components. These components are arranged into an architecture that supports the execution of cross-domain workflows by assigning specific roles to each of these components. Deliverable D2.2 provided a technological overview of all the key technologies that will be included in the ACROSS platform. Included to these, software technologies being part of the ACROSS orchestrator have been discussed. Additionally, the D2.2 report provides a comprehensive description of each key technology and identified lack of how such technologies will be integrated together. This latter aspect is within the scope of this document. As such, the analysis of the system requirements allows us to devise these key technological components organized in three main architectural levels (it is worth to note that these components will be extended during the project timeframe or they will be developed from scratch in case they are completely new), as described in the following:

- *(High-level) workflow execution engine*: cross-stack workflows should be defined with a clear identification of the control/data dependencies among the various steps (in the form of a Directed Acyclic Graph –DAG) and the identification of the proper execution environment for each step. To this end, **StreamFlow** provides the necessary capability and expressiveness for describing and executing (complex) workflow's DAG by easily matching with proper execution environments.
- *Execution environment deployer*: to match the DAG steps with the specific execution environments it is necessary to deploy these environments on the computing resources (both Cloud and HPC). While the installation of the software packages required on the HPC side is done by system administrators, **Ystia** is the selected tool for that purpose on the Cloud side. This solution allows to describe all the necessary deployment steps by chaining abstract blocks (application templates) that identify the specific resources to match (e.g., a compute node with at least 16-cores and 32GB of memory) and specific software to install (e.g., installing Singularity container engine on top of a virtual machine). Ystia is composed of two main modules: *i*) the front-end named **Alien4Cloud** (A4C) which can access a catalog (i.e., a repository) of abstract blocks and application templates; *ii*) the back-end engine named **YORC**, which executes the deployment steps. Some of the ACROSS Pilot applications rely on specific software components for executing the workflows, and in some case such (low-level) components can run on Cloud resources. In these cases, Ystia will be responsible for installing these components as part of the plan for creating the execution environment on the Cloud.
- *Advanced resource reservation system*: as stated in the Section 1 and Section 2, some limitations seen in current HPC systems are found in the workflow-unawareness and limited awareness of HW heterogeneity coming from the batch schedulers. To overcome the first limitation, an advanced resource reservation and job submission schema should be used, and this is the purpose of the **WARP** (Workflow-aware Advanced Resource Planner) module. It will leverage the advanced reservation features exposed by current-generation batch schedulers to properly reserve (in advance) specific sets of HPC resources.
- *(Low-level) job execution on heterogeneous environments*: exploiting the large heterogeneity of computing resources (GPUs, different types of CPUs – AMD Epyc and Intel Xeon, FPGAs, etc.) requires optimizing the way job's computations are distributed on the computing resources. A finer grain control of the underlying resources is thus required, and this is the purpose of the **HyperQueue** tool. It allows scheduling tasks contained in the jobs submitted to the batch scheduler with high flexibility and without losing performance. The basic idea behind it is to have a master agent (still exposing queues) on the HPC frontend (actually, on the login nodes or on a virtual machine hosted in the Cloud partition) and a set of worker agents that can directly communicate with the master to coordinate the execution of tasks (and without waiting long times due to batch scheduler queues). Worker agents can also be easily installed on virtualized resources as provided by the Cloud partitions.

Figure 3 depicts these components and traces their relative interactions (control/data paths). Besides these core components, an additional module has been envisioned to better address the modelling and execution (training) of ML/DL workflow steps. This component, named **FMLE** (Fast Machine Learning Engine) provides a high-level interface for designing, training, and deploying ML/DL models as it leverages a direct connection with A4C to generate specific application templates and execute them through the YORC engine.

As Figure 3 shows, there are multiple interaction paths (arrows with different colors) that are activated at different points in time. The first stage is represented by the (asynchronous) submission of the workflows to the orchestrator (gray arrows) or the starting of an interactive session (this will mainly use Cloud resources) for ML/DL training and algorithms tuning. Whenever a workflow is submitted it is enqueued by WARP, i.e., all the information required to properly execute the workflows are stored in the WARP datastore. This allows the

WARP module to have visibility on multiple workflows to run and their requested resources. The second stage is represented by the HPC resources reservation in advance (on the other hand, Cloud resources can be demanded at runtime). Current-generation batch schedulers (SLURM, PBSpro) allow users to request the use of HPC resources for a future point in time (green arrows in Figure 1 represent the first phase of reserving resources and notification to StreamFlow). Once that point is reached, the user that requested the resources can submit jobs to the batch scheduler without incurring in queue waiting times, since the job submission can be done by specifying the reservation. This mechanism also preserves priorities, since the batch scheduler may agree or not upon the request. Once the resource becomes available, StreamFlow will be informed, and it will be able to start the execution of the specific workflow (or portion). At this stage, StreamFlow gets access to the workflow description and the execution environment needed by the workflows to run.

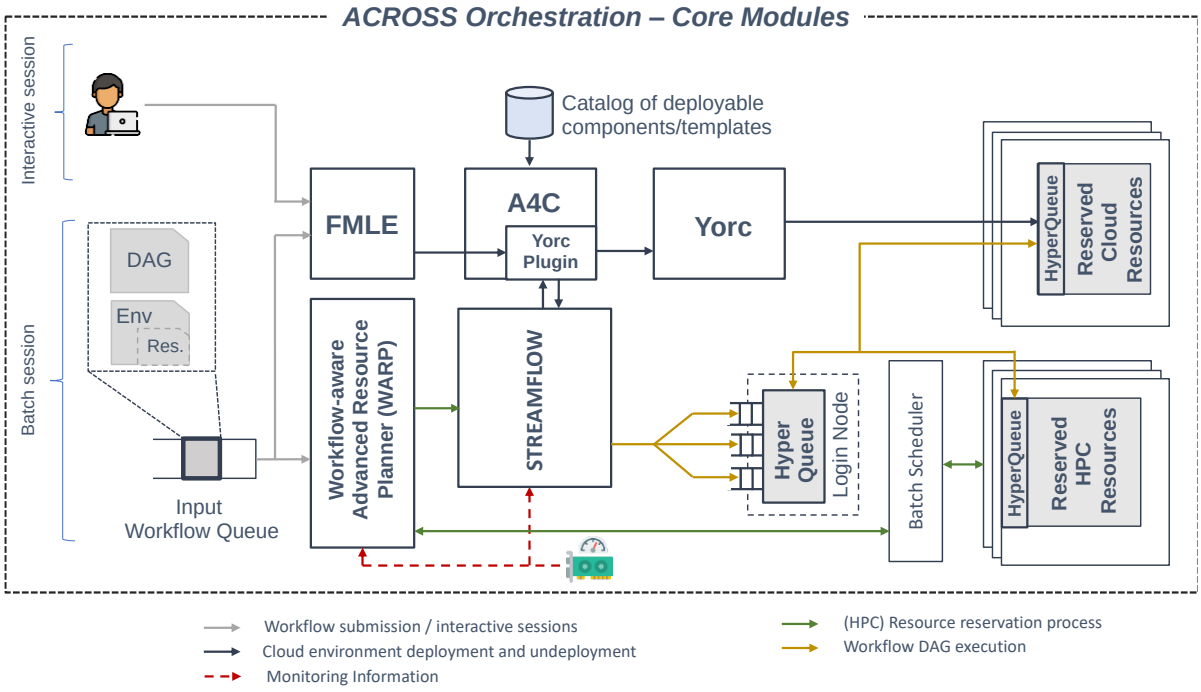

Figure 3 – Core components of the ACROSS Orchestration solution.

The blue arrows in Figure 3 provide control/data paths used by StreamFlow to ask A4C and YORC to prepare the necessary execution environment on the Cloud partition; they can also be used to configure HyperQueue workers on the Cloud resources and make them connecting to the HyperQueue server on the HPC login node). Once the execution environment is created, the workflow execution can start (yellow arrows). Here, StreamFlow coordinates the execution, and through the HyperQueue it will be possible to optimally use heterogeneous resources. Red dashed arrows provide the path for gathering information coming from cluster monitoring systems, which could be crucial for making optimal orchestration decisions in terms of energy efficiency and performance.

3.1 Integration with external computing platforms

As the ACROSS platform is focused on performance and energy-efficiency, fulfilling with other 'platform integration requirements', such as that of providing a secure access to the computing and storage resources, can be achieved by integrating the core structure with other additional modules. It is also important to note that these additional modules are not strictly required to support the execution of the workflows as we already described; however, they become important to fulfill these additional requirements and to match with specific policies of the HPC centers. Another aspect to consider is that these additional modules are already available within other 'external' computing platforms. Thus, defining a clear mechanism to connect the ACROSS orchestration core components with these external platforms, is of help in projecting the ACROSS platform to support the execution of a broader set of applications, and thus not being restricted to only ACROSS pilots.

Supercomputing resources are becoming even more attractive to a large set of end users and research communities; as a matter of fact, providing a mechanism to better control the access to those resources becomes of paramount importance. More than security aspects, also the capability of distributing the computations among resources located on multiple sites is gaining momentum. As an example of this, Urgent Computing applications may benefit from such distribution of resources. In fact, the workflow is generally

executed in such a way the chance of completing the computation within a time boundary is maximized. Another case is given by those applications that require a high level of redundancy in their execution (e.g., weather forecasts). Although such support is not mandatory in the ACROSS project, allowing the ACROSS orchestrator to be extended and integrated with another system (e.g., the LEXIS platform) provides an effective solution for all these cases.

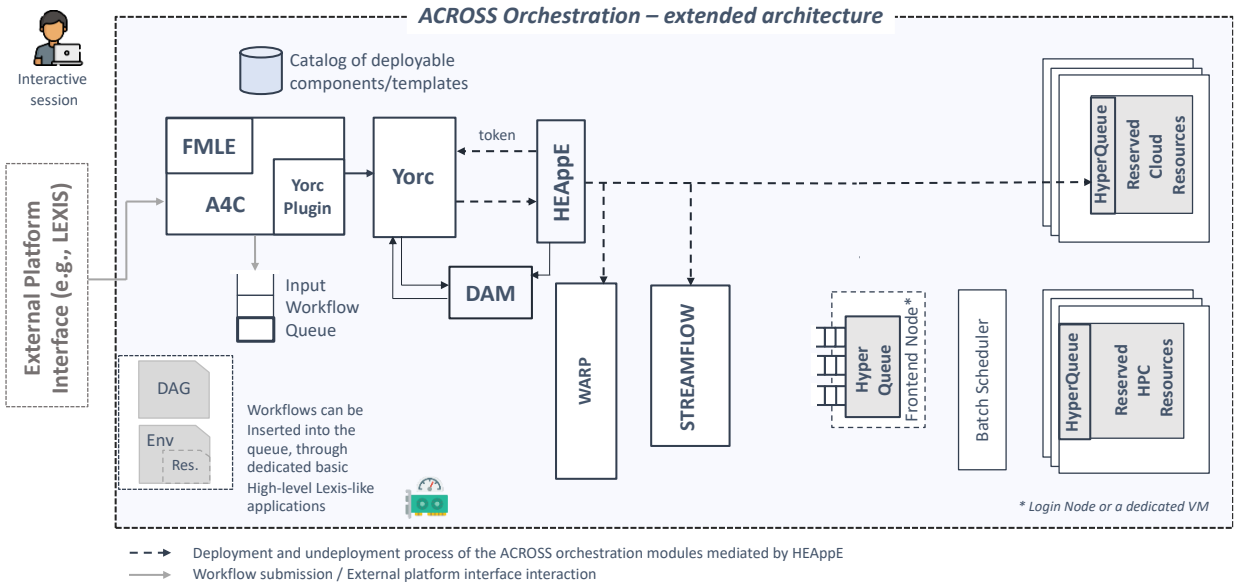

Figure 4 – Extended ACROSS Orchestrator architecture coupling with an external (meta)orchestrator (e.g., LEXIS platform orchestrator).

As an example of how this integration can be achieved, Figure 4 shows a possible instantiation of the ACROSS orchestrator (core components) along with other specific modules, namely HEAppE and the DAM. The former is a middleware solution developed by IT4Innovations to provide an HPC-as-a-Service solution. Its main purpose is to decouple the users from the cluster accounts that can launch and execute jobs on the HPC resources. This is done by keeping an internal dynamic mapping between the users (i.e., portal users) and HPC accounts on the HPC machines. It is also capable of providing authorization tokens for getting access to the Cloud resources. The latter (Dynamic Allocation Module) is in charge of dynamically selecting the best location (i.e., cluster for any available computing center) where to execute a given job, based on the available resources they expose, planned maintenance periods, and resources requested by the job.

3.2 Envisioning integration with neuromorphic devices

Fulfilling the performance and energy efficiency requirements is done through a broader usage of HW accelerators, and through an effective mechanism to control such heterogenous resources with a fine granularity. While GPUs and other specialized HW devices offer high tradeoff between performance and power consumption with regards to CPUs, the demand for even more specialized devices remains. The ACROSS project foresees the investigation of innovative architectures, and more specifically the porting of conventional artificial neural network (ANN) models to *neuromorphic* devices. The latter have been emerged as computing systems able to mimic the behavior of their biological counterpart closely than conventional neural network models. As such, Spiking Neural Networks (SNNs) show many advantages [9], especially if the execution of their models is done on specific neuro-inspired chip architectures. Among the others, such advantages include greater energy efficiency, capability of modeling dynamical modes of network operations, computing in the continuous real-time domain, etc.

Supporting this type of HW architectures requires to set up and integrate, in the overall platform, different components (both HW and SW). Some of the software components must be properly addressed by the ACROSS orchestration system. Indeed, ACROSS partners found that the most viable way of experimenting with neuromorphic devices is: *i*) emulating the hardware specialized architecture as an overlay architecture running on top of a FPGA fabric; *ii*) porting a pre-trained ANN model on the SNN emulated device, by translating an intermediate representation of the ANN model to a SNN. This latter step requires to pre-train the ANN model using more conventional systems (e.g., GPUs or NNPs), which results into an intermediate representation, and then to translate this latter into the equivalent SNN by using specific tools (compilers).

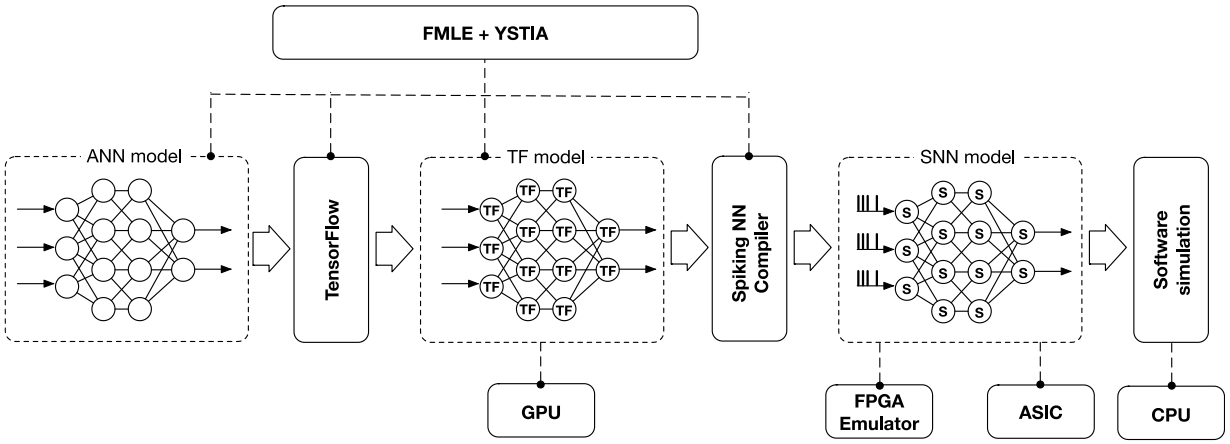

Figure 5 - Process of translating a conventional ANN model into the equivalent SNN through the FMLE/YSTIA toolchain. The execution of the SNN can target SW emulation, specifically ASIC or FPGA emulation.

Training the ANN model involves tuning hyperparameters of the models (i.e., weights, how the layers relate to the others, etc.). This phase generally requires using a large set of computing resources; as such, the underlying frameworks (e.g., TensorFlow, PyTorch, etc.) expose convenient mechanisms to accomplish this task. All these frameworks have been developed to abstract and hide the complexity of the mechanics that make ANN models seamlessly running across different configurations of the computing infrastructures. The ACROSS project aims at further easing their management and move the abstraction process even at a higher level. To this end, the ACROSS orchestrator architecture integrates a specific module, namely Fast ML Engine (FMLE). By relying on this module, will be possible to easily describe an ANN model, tuning and training it using conventional infrastructural resources (GPUs, NNPs), and finally compiling it into an equivalent SNN model. At the time of writing this document, the ACROSS partners identified as a viable solution the implementation of the ANN model using the TensorFlow framework, and its compilation into a SNN using an ad-hoc compiler [10]. Figure 5 shows a graphical representation of this process.

4 Conclusions

The ACROSS project targets the efficient execution of cross-domain workflows, by exploiting the large HW heterogeneity brought in the project (both in terms of HW accelerators and different CPU architectures) and by improving the mechanism for allocating computing resources and scheduling the jobs on top of those (orchestration). In this regard, WP4 is focused on this second aspect, which involves the design and implementation of a workflow orchestrator. This implies to gather and analyse all the requirements that are important to drive the decision-making process, i.e., deciding which software components are needed to accomplish the purposes of the orchestrator and how they connect with each other.

The purpose of this document is collecting all these requirements (here generically identified as 'system' requirements) and analysing them to derive the collection of orchestrator SW components and their specific interaction. To this end, the document categorises the system requirements in 4 classes (platform integration, workflow specific, pilot specific, performance and energy efficiency): for each requirement category the way ACROSS orchestrator intends to fulfill the requirements is provided. Indeed, given all these requirements, an updated vision of the ACROSS orchestrator architecture is provided. This document also traces a path towards the integration with other external platforms and the integration and management of neuromorphic chips. Although the architecture is described at a fine granularity, it still provides a high-level overview of the ACROSS orchestrator, leaving the details of various modules and their internal logic to future WP4 deliverables.

References

[1] A. Scionti, J. Martinovic, O. Terzo, E. Walter, M. Levrier, S. Hachinger, D. Magarielli, T. Goubier, S. Louise, A. Parodi, S. Murphy, C. D'Amico, S. Ciccia, E. Danovaro and M. Lagasio, et al., "High performance computing (HPC), Cloud and Big-Data Convergent Architectures: The LEXIS Approach HPC, Cloud and Big-Data Convergent Architectures: The LEXIS Approach," in *The 13th International Conference on Complex, Intelligent and Software Intensive Systems (CISIS 2019)*, 2019.

[2] D. H. Ahn, N. Bass, A. Chu, J. Garlick, M. Grondona, S. Herbein, H. I. Ingólfsso and M. Tauffer, "Flux: Overcoming scheduling challenges for exascale workflows," *Future Generation Computer Systems.*

[3] M. A. Salim, T. D. Uram, J. T. Childers, P. Balaprakash, V. Vishwanath and M. E. Papka, "Balsam: Automated Scheduling and Execution of Dynamic, Data-Intensive HPC Workflows," *ArXiv,* 2018.

[4] A. Al-Saadi, D. H. Ahn, Y. Babuji, K. Chard, J. Corbett, M. Hategan, S. Herbein, S. Jha, D. Laney, A. Merzky, T. Munson, M. Salim, M. Titov, M. Turilli and J. M. Wozniak, "ExaWorks: Workflows for Exascale".

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

[6] "Weather Research and Forecasting model," [Online]. Available: https://www.mmm.ucar.edu/weather-research-and-forecasting-model.

[7] M. Dorier, G. Antoniu, F. Cappello, M. Snir, R. Sisneros, O. Yildiz, S. Ibrahim, T. Peterka and L. Orf, "Damaris: Addressing Performance Variability in Data Management for Post-Petascale Simulations," *ACM Transactions on Parallel Computing (ToPC),* vol. 3, no. 3, 2016.

[8] "ERT - Ensemble based Reservoir Tool," [Online]. Available: https://github.com/equinor/ert.

[9] D. Nekhaev and V. Demin, "Recurrent Spiking Neural Network Learning Based on a Competitive Maximization of Neuronal Activity," *Frontiers in Neuroinformatics,* vol. 12, no. 78.

[10] "SNN toolbox," [Online]. Available: https://snntoolbox.readthedocs.io/en/latest/guide/intro.html.