ACROSS

HPC Big DAta ArtifiCial intelligence cross Stack PlatfoRm TOwardS ExaScale

D4.3 – Analysis and definition of scheduling strategies for resource optimization

Deliverable ID	D4.3
Deliverable Title	Analysis and definition of scheduling strategies for resource optimization
Work Package	WP4
Dissemination Level	PUBLIC
Version	0.6
Date	2022 – 08 – 31
Status	FINAL
Deliverable Leader	LINKS
Main Contributors	Scionti A. (LINKS), Vercellino C. (LINKS), BULL, IT4I

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	2022-06-10	LINKS Foundation	ToC (Draft)
0.2	2022-07-06	LINKS Foundation	First contributions to the first chapter
0.3	2022-07-11	LINKS Foundation	Contribution to the second chapter
0.4	2022-07-18	LINKS Foundation	Contribution to second and third chapters
0.5	2022-07-22	LINKS Foundation	Finalizing the document
0.6	2022-08-01	LINKS Foundation	Internal review comments have been addressed

Table of Contents

Glossary

Acronym	Explanation
ANN	Artificial Neural Network
CLI	Command Line Interface
CPU	Central Processing Unit
CWL	Common Workflow Language
DL	Deep Learning
FPGA	Field Programmable Gate Array

FMLE	Fast Machine Learning Engine
GA	Genetic Algorithm
GPU	Graphic Processing Unit
HPC	High-Performance Computing
HPDA	High-Performance Data Analytics
ILP	Integer Linear Program
ML	Machine Learning
SH	Schedule Horizon
SNN	Spiking Neural Network
SotA	State-of-the-Art
TOSCA	Topology and Orchestration Specification for Cloud Applications
UI	User Interface
WARP	Workflow-aware Advanced Resource Planner
WAS	Workflow-Aware Scheduling
WP	Work Package

List of Figures

List of Tables

Executive Summary

The ACROSS project relies on the integration of various sophisticated hardware and software technologies into a platform that enables efficient execution of workflows composed of steps involving different types of operations (i.e., large numerical simulations, machine learning and deep learning tasks, HPDA, etc). Thus, energy efficiency and performance are achieved, on the one hand, through the large adoption of hardware accelerators (different accelerators can be targeted depending on the workflow step, e.g., GPUs, FPGAs, etc.). On the other hand, the 'effective' execution of such kind of workflows on current and upcoming supercomputers is closely related to the way the underlying management software operate and more specifically on the implemented and applied resource management and workload scheduling strategies.

This deliverable focuses on the *analysis and definition of scheduling strategies for resource optimization* that are under development in ACROSS. In the ACROSS project, different HPC infrastructures are accessible, each of which has its peculiarities that must be captured and translated into specific resource management strategies, which is one of the core elements of the ACROSS orchestrator. Nevertheless, the ACROSS orchestrator must be flexible enough to adapt to all these situations without requiring major changes to its internal logic.

For this reason, this deliverable aims at thoroughly illustrating the specific strategies and related software technologies underlying the resource management and workload distribution. As such, the document begins with an overview of some of the limitations we see in the State-of-the-Art batch scheduling systems. This is followed by the description of what the ACROSS multi-level orchestrator can do to overcome these limitations, i.e., enabling a *deterministic execution of workflows* and possibly *applying optimal scheduling to jobs* belonging to separate workflows (even if they belong to the same computational project). This clear statement of the problem is then translated as a formal description of the problem, along with the proposed solution. The description of the main components (WARP, HyperQueue, FMLE/Ystia) involved in resource management is given.

Position of the deliverable in the whole project context

This deliverable covers technical aspects related to the management of the compute resources that are accessible in the context of the ACROSS project, and specifically how these resources can be reserved (in advance) and how the reserved resources can be filled in with workflows' jobs in an optimal way. To this end, the deliverable restricts its focus to a specific set of components of the orchestrator architecture, which are the ones closely involved in the resource reservation process, the planning of execution of workflows and the management of the acquired resources with a fine granularity. As such, this deliverable covers the activities carried out in WP4 and specifically related to T4.3. Nevertheless, the strategies described in this document have a close relation with the activity carried out in WP3, since the management of resources with a fine granularity also involves the access to accelerators. Also, the activity related to the investigation of neuromorphic simulation is linked: FMLE has been selected as the main tool for driving the training of Deep Learning models (ANNs), possibly building a pipeline that includes their conversion to Spiking Neural Networks (SNNs) to support the execution on neuromorphic hardware; likewise, Ystia has been selected as the main tool for the deployment of all the necessary software on the cloud. Connection with WP2 and WP3 is seen, as the WARP module and its interaction with the other orchestrator components is part of the overall co-design approach. Pilots (WP5, WP6 and WP7) provided useful insights of their workflows and, as such, this information has driven some of the choices made for the resource management strategy implemented by the WARP in conjunction with StreamFlow and HyperQueue.

Description of the deliverable

This deliverable is organized as follows. An introductory chapter is devoted to providing the context, i.e., supercomputer resources are today managed through State-of-the-Art (SotA) batch scheduler systems, which are configured by system administrators to implement specific access policies. As such, the batch scheduler internal logic is optimized to maximize resource usage and to guarantee a fair access to all the users. Given this, the focus of this document is the analysis and definition of scheduling strategies for resource optimization, with chapter two starts stating the main limitations that batch scheduler systems are subject to. With some concrete examples, the chapter expands the discussion by formalizing the resource management problem as it is considered in ACROSS. The identified strategy to overcome the presented limitations is implemented within the logic of the WARP module (see *D4.1 – System Requirements Analysis for Orchestrator Design* [9], *D2.2 – Description of key technologies and platform design* [10]), thus, part of the document is devoted to describe the main architecture of the WARP module, the interaction between its internal components and the other orchestrator components. The third chapter provides information concerning the way HPC compute resources acquired by the WARP module can be managed with a fine granularity. Similarly, the chapter describes how cloud resources are managed by means of Ystia to enable FMLE integration and to support the eventual execution of pre-, post-processing and visualization steps that do not require the use of HPC cluster's resources.

1 Introduction

Applications and their associated workflows are becoming more complex than in the past, often involving processing steps related to machine learning (ML), deep learning (DL), or high-performance data analytics (HPDA). These workflows also combine these types of steps within iterative loops with some specific dependencies on each other, and sometime require jobs to run longer than the queue to which the jobs are submitted allows. Job scheduling and compute resource allocation in the HPC domain is done by means of State-of-the-Art (SotA) batch schedulers. HPC system administrators are able to configure batch schedulers to consider the diversity of resources available on the machines and to tune the scheduling policies in such a way so that priorities match those required by their business, fair access to the resources is still guaranteed and the resource utilization is maximized. Although, batch schedulers can be configured in very sophisticated ways to fulfil such requirements, the configuration is generally done at the beginning, while the requirements may change over the time. Also, while scheduling algorithms can take into account job dependencies, their visibility is restricted to the level of the single jobs, thus limiting the search space for optimizations. Whenever the submitted jobs take more time than the maximum allotted for a certain queue, jobs must be checkpointed and restarted. In this context being able to reduce as much as possible the impact of queuing time may provide an improvement over the workflow execution.

With this in mind, being able to optimize the execution of workflows by providing a mechanism for their more deterministic execution represents an important objective in the context of WP4. Considering system requirements (see D4.1 [9]), resource optimization passes through the capability of controlling the allocation of resources with a granularity that is below that of the single nodes; in other words, being able to assign a fraction of the node's resources to a job and the remaining to another enables the orchestrator to fulfil the co-location requirement.

To this purpose, the ACROSS orchestration architecture (as described in D4.1 [9]) includes a set of components whose main objective is that of *i*) reserving the compute resources needed to run the jobs with a degree of visibility that spans the set of workflows belonging to the same computational project (i.e., the whole set of resources that are associated to a given group of users); *ii*) planning the execution of jobs on top of the reserved resources to provide (a more) deterministic execution of the workflows; *iii*) controlling the resource allocation in such a way that the resources of a single node can be fractioned and nodes can be assigned to different jobs concurrently. Finally, *iv*) cloud resources can be accessed to execute specific parts of the workflows, taking the benefit of virtualization. In the following chapters, all these aspects will be covered, along with the description of the architecture of WARP, which is the module responsible for making advance resource reservations.

It is worth to note, that the mechanisms behind the implementation of WARP aim at not breaking the scheduling policy implemented by the batch schedulers, and at not negatively affecting the priority levels already set in the queues.

1.1 Scope

The scope of this document is diverse, although it focuses on aspects related to the way computing resources are acquired and managed. In particular, we analyse some of the limitations we have encountered when relying only on SotA batch schedulers to improve the execution of workflows belonging to the same computational project. While batch schedulers can maximize resource utilization and ensure equitable distribution among users, their visibility is still limited to the job level. Also, the execution of jobs whose maximum clock wall time exceed that allotted by the submission queue can benefit from an improved approach for acquiring compute resources. Once the problem is stated, a more formal formulation follows. This latter serves as the background for defining and implementing the resource allocation logic within the WARP module along with the mechanism for reserving resources in advance. As such, the main architecture devised for the WARP module is presented. Scheduling strategies for resource optimization are not restricted to the approaches for getting access to the compute resources but also involve those mechanisms that allow to optimally distribute the workload on those resources. To this end, HyperQueue provides the necessary machinery to 'assign' the execution of jobs (and consequently assigning tasks contained in the jobs) on the reserved resources, with a granularity that is lower than that of an entire node. These mechanisms are discussed too. Similarly, the document describes how Ystia tools are used to acquire and manage cloud resources, as they are exploited by FMLE and workflows.

1.2 Related documents

ID	Title	Reference	Version	Date
[RD.1]	Description of key technologies and platform design	D.2.2	0.8	30-11-2021
[RD.2]	System Requirements Analysis for Orchestrator Design	D.4.1	0.7	28-02-2022

2 Workflow-aware Optimization Strategies

Workflows are a convenient way to describe a series of steps that must be performed as part of the execution of an application; as such, these steps may have (data) dependencies and execution relationships that can be conveniently expressed as a graph structure. Nodes represents the steps to be executed, while the edges represent the relationships/dependencies between the nodes. In most cases, execution steps map directly to the *submission* of a job to the HPC execution infrastructure, so optimizing the execution of these jobs seems to be reasonable as applications can gain benefits in terms of reducing the overall execution time. The submission process is managed by the HPC batch scheduler, which has two tasks: *i*) maximizing the use of machine resources (i.e., trying to saturate the machine resources by running as many independent jobs as possible); *ii*) scheduling jobs in such a way that satisfies queues' priorities and enforced scheduling policies are fulfilled.

State-of-the-Art (SotA) batch schedulers apply very sophisticated algorithms to ensure resource usage maximization and guaranteeing the respect of the queues' priorities and scheduling policies. These latter are generally predefined at the beginning by system administrators considering specific conditions of the machine, which may result in a mismatch with conditions over the time. Besides these limitations, other factors have been found in ACROSS to negatively impact on the execution of workflows. Despite batch schedulers configurations allowing the consideration of job dependencies to optimize their scheduling, other factors influence the overall execution. As such, the queuing time that each submitted job experiences is *nondeterministic* over the entire execution of the workflows; indeed, the queuing times strongly vary with the number and type of already submitted jobs, with the resulting times being very difficult to learn. The ACROSS co-design activity has allowed us to devise a solution to cope with this limitation, without the need of implementing complex strategies requiring the collection of large historical data sets, while leveraging on advanced features already exposed by SotA batch schedulers.
The proposed solution has been found to provide advantages also in other contexts in which the queuing time becomes the major execution limiting factor. Indeed, queues are associated, among the others, to a maximum allotted time (i.e., maximum wall clock time) for the execution of a job. This latter implies that jobs whose execution exceeds the maximum wall clock time are stopped. Thus, to guarantee the proper execution, such kind of jobs must implement sophisticated checkpoint and recovery strategies, i.e., a snapshot of the status of the job is taken before the job is stopped and then it is used to restart the job in a later point in time; although this process guarantees the correctness in the job execution it does not avoid the job to incur time in being queued.

The ACROSS solution relies on a sophisticated mechanism to acquire the HPC execution resources in advance which then allows planning the execution of the jobs by considering their specific data dependencies and relationships. In the following, details are provided concerning the process of resource acquisition and the way ACROSS orchestrator tries to optimally schedule jobs on top of them.

2.1 Main Problem Statement and Analysis

As stated before, SotA batch schedulers (e.g., SLURM, Altair PBSpro, etc.) are configured in such a way their visibility over the workflows is (generally) confined to that of the single jobs. Although it is possible to define dependencies among them, visibility over the entire workflow or even among multiple workflows represents a non-common setup. In ACROSS, we claim that this kind of visibility level is of worth to improve the execution of workflows by limiting or even better avoiding the impact of the jobs' queuing time on the overall execution.

Figure 1 depicts the typical situation where resources are assigned to *running* jobs in such a way their usage is maximized over the time, and where resource assignment is also planned for jobs waiting in the queues (*scheduled* jobs) depending on the specific priorities and scheduling policies enforced by the system administrators. As the reader can easily realize, the batch scheduler keeps track of the jobs that are executing (along with their status) and those that are waiting in the queues; based on the enforced priorities and scheduling policies, it also creates a *schedule* for the jobs currently waiting in the queues. This translates into assigning computing resources to these jobs, while the schedule is periodically updated to reflect the status of queues.

In this situation, it makes sense to imagine querying the batch scheduler and retrieving the information that allows us to determine which are the last scheduled jobs. Also, based on the information carried out by the jobs concerning their maximum wall clock time, it becomes possible to combine this information and determine a point in time in the future, where it is reasonable to assume that no one job is currently scheduled (or is far enough in time that affecting the current schedule is deemed acceptable). We define this point in time as the *schedule horizon* (SH).

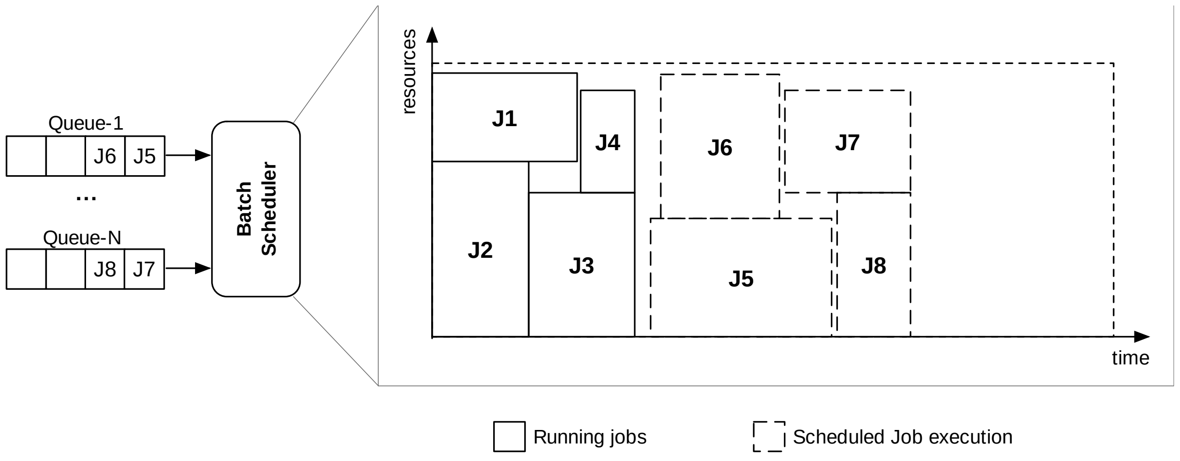

Figure 1 - Typical situation for scheduling jobs on available resources

Thus, the SH defines the temporal boundary, starting from which, it is possible to reserve (in advance) compute resources, which are later assigned to specific jobs. The advantage offered by this approach, is that it minimizes the interference with the batch scheduler logic; furthermore, it does not require any change to the scheduling policies and queue priorities. Figure 2 depicts the same situation of Figure 1 with the SH and reserved resources highlighted.

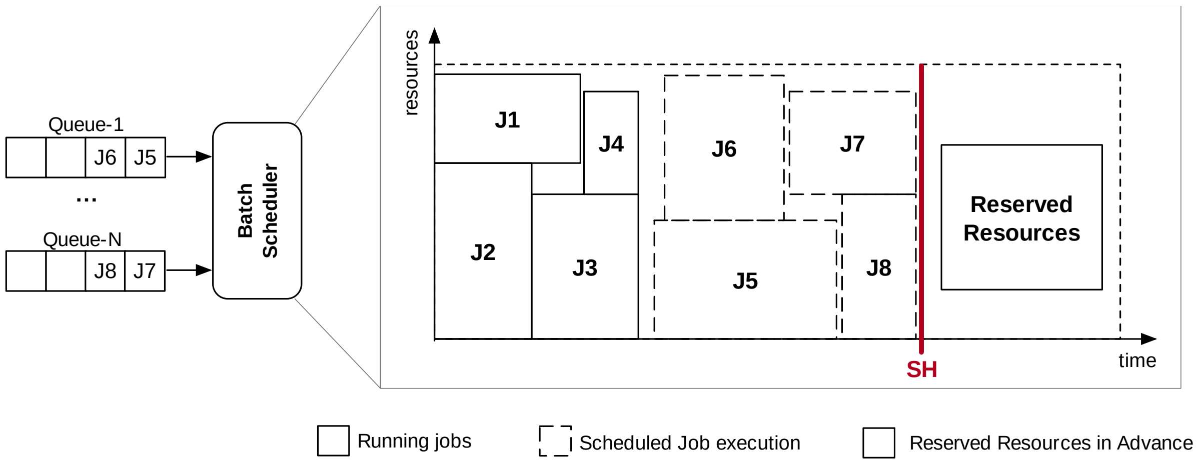

Figure 2 - Identification of the Schedule Horizon (SH) and reservation of computational resources in advance

Worth to mention is the fact that the proposed approach has been devised to be an alternative management model with respect to the traditional batch submission model; as such, we see the applicability of the proposed management model to those cases where the workflows are composed of (heterogeneous) steps with dependencies and where more guarantees regarding the overall duration of the workflow execution are needed. Moreover, our approach is intended to facilitate the achievement of three important features, i.e., the co-scheduling of jobs that needs to communicate each other but reside on different queues/locations, the co-location of jobs that share resources (i.e., the resources at the node level can be fractured and assigned to different jobs with a granularity that is finer than that of the single node) and to reduce the start-up time of checkpointed jobs. Conversely, we see the traditional batch scheduling approach well suited for all those cases where the application is less sensitive to the waiting time of queued jobs.

2.2 Workflow-aware scheduling strategies

The proposed *workflow-aware scheduling* (WAS) strategy is based on two main pillars: on one side, the capability of making resource reservations in advance and, on the other hand, to assign those reserved resources to jobs also belonging to different workflows. As such, the WAS strategy aims at reducing the overall execution time for completing a workflow (deterministic execution) and maximizing the resource usage. The former is fulfilled by planning the execution of jobs belonging to the same workflow as closer in time as possible, still preserving dependencies. The latter is fulfilled by defining the problem as a combinatorial optimization problem that is solved periodically to provide the optimal job allocation. As such, the reservations are treated as bins, while the jobs to execute are the objects to use filling the bins. Possibly, the formulation of the combinatorial optimization problem can be as such that both the above mentioned requirements are fulfilled by feasible solutions.

While the proposed approach is general enough to consider the planning of job execution over reserved resources for workflows belonging to different groups of users, for simplicity, we assume to restrict the assignment of resource reservations to jobs belonging to workflows which are part of the same computational project. As such, the reservation and allocation machinery are simplified, since it can operate under the assumption that the owners of the jobs planned for the execution are all authorized to get access to the same pool of resources. Indeed, a computational project defines the overall amount of compute resources (generally expressed as core hours) that a certain group of users can consume for running their jobs/workflows.

2.2.1 Problem Statement

The statement of the problem starts from the identification of some requirements and assumptions we need to make to properly devise the strategy for making reservation of the resources.

The first thing to note is that the physical domains involved in the resource allocation process (time, amount of compute resources required –i.e., cores, amount of memory required) must be discretized. This implies the ability to define a kind of unitary values for such domains. Concerning the time unit, the discretization can be done considering a temporal window in the order of tens of minutes. On the other hand, the minimum compute resource that can be allocated is the single core of a CPU. The memory unit can be derived considering that the entire memory pool of a node is uniformly distributed to all the available cores. Another point to consider is the heterogeneity of the resources, i.e., the availability of hardware accelerators within the nodes. In this case, the simplest assumption is that the minimum compute resource that can be allocated is the single accelerator within the node (i.e., the single GPU card, the single FPGA card, etc.).

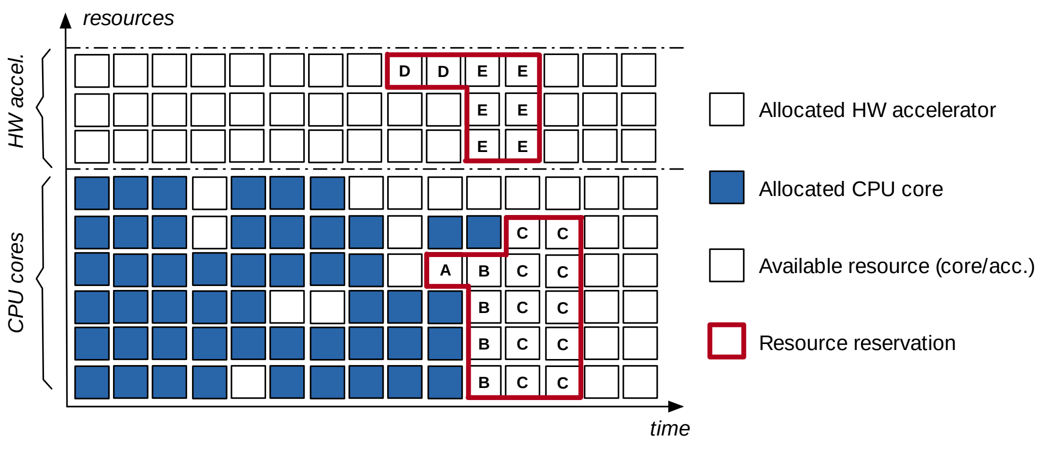

Figure 3 - Example of the allocated resources over time (resources are discretized) and reservations made for CPU and HW accelerators

The problem of finding the best resource reservation is reverted into an optimization model that will be solved online. In [1][2] the authors modelled the scheduling problem as an optimization problem solved using a heuristic (genetic algorithm –GA). The authors suggested that the scheduling strategy should respond in a range of 15sec - 30sec to 1min, so we will tune our reservation strategy to operate in a time window of few minutes. The complexity of finding a proper allocation of resources (which is based on making a reservation for a future point in time) strictly depends on the dimension of the search space for a solution. This in turn depends on the physical domains we decide to consider; as such, to simplify the problem, in a first instance we assume that the required memory to make the reservation is set to the maximum available on the nodes. Given this, the number of dimensions (and thus of variables in the associated optimization problem) to treat is reduced. Nevertheless, this assumption is valid in general, being potentially not optimal from the perspective of the memory allocation. To solve this problem, a subsequent optimization step can be performed, to determine if a configuration of the reservation requiring less memory is available. Figure 3 depicts a typical situation where the coloured squares represent the already allocated resources (discretized in the time and resource –cores– dimensions). Despite the long-term goal will be that of optimizing the execution of the whole set of workflows submitted within a given computational project, in a first phase we focused on an approach where we target to optimize the execution of a given workflow. Given that, each workflow belonging to the same computational project will be treated independently from the others.

The objective of the resource reservation strategy is that of finding the optimal shape of the reservations in such a way the time required to complete the execution of a given workflow is minimized (the red boxes in Figure 3). To this end, it is reasonable to allow the resource allocation strategy to combine different reservations to create the shape. For instance, in Figure 3, the reservation for the CPU cores may combine three subsequent reservations identified with A, B and C labels. Similarly, for the GPUs, combining reservations D and E allows the system to create the optimal shape. To pursue this overall goal, it seems reasonable to investigate a *two-step optimization approach*: dividing the problem in two phases will allow to reduce the dimensionality of the problem and eventually even its complexity in terms of computational time to retrieve feasible solutions. To be more precise the two identified steps consist of:

- **STEP-1.** Sort the list of submitted workflows (within a properly defined time window) and select the first in the sorted list. Given the workflow, extract the minimal graph representing the execution steps of the workflow; to this end, it is worth to note that in realistic workflows (as those defined within the ACROSS project) some of the operations to be performed can be aggregated in one single step in the graph. The resulting graph has for each node the requirements in terms of CPU cores and amount of memory (although, to simplify the initial finding of a feasible solution, this latter is assumed equals to the maximum available in the nodes). Furthermore, such representation, reasonably, is obtained from StreamFlow [14], which provides features to extract an internal representation from the input CWL description (see D4.1 [9]). STEP-1 strategy will advocate for the definition of a sorting policy, which will consider workflows' arrivals time, resources' requirements of the jobs and priority scores, a particular attention will be paid to avoid unfair scheduling and workflows starvation.

- **STEP-2.** Given the graph representation for the selected workflow, finding a (feasible) optimal allocation of the compute resources that minimises its overall execution time (i.e., the makespan), considering the expected status of the resources that is retrieved through the batch scheduler. STEP-2 will take advantage of the deterministic/fixed information coming from STEP-1 to model the optimization problem for the actual resources' reservation. Solving the optimization problem will be achieved through the implementation of a solver function (based on a meta-heuristic [1][3][4][5], traditional optimization process, or a combination of different techniques) –i.e., the optimization phase. For the sake of the correct problem modelling, the envisioned optimization phase will consider only those supercomputer partitions whose capability is equal or larger to that required to execute the steps of the workflow; specifically, the memory amount will be assumed to be the maximum available on the nodes. As such, this simplifies the optimization problem definition by reducing the number of variables (indeed the variable used to model the memory can be avoided). In a subsequent step, the memory usage can be further optimized by selecting, among the N best feasible solutions, the one that also minimizes the allocated memory per node.

Willing to formalize the above-described approach, we can define the programming optimization problem; as such we can define the following entities:

- $\mathcal{G}(\upsilon, \varepsilon)$: the graph representing the reduced workflow coming from the sorted list from STEP-1, the set of nodes υ represents the blocks of resources to be allocated (bear in mind that even if the blocks are managed individually from the optimization point of view, the actual resource reservation can reserve more than one block at a time); $\forall i \in \upsilon$ it is known the amount of resource required (i.e., r_i), if there are any specific hardware requirements (i.e., GPU, FPGA, etc.), and the execution wall time w_i. The set of directed edges ε defines the allocation order: $\mathcal{N}_i^+ = \{j \in \upsilon : (i \rightarrow j) \in \varepsilon\}$ describes, for each $i \in \upsilon$, the execution blocks that must be allocated after i.

- $\mathcal{T}$: is the set of all the timeslots available across all the resources.

- $\mathcal{R}$: is set of resources.

To formalize the problem, the variables reported in Table 1 are used.

Table 1 - Variables of the optimization programming model related to the allocation of compute resources (reservations).

Variable	Description
x_{ij}^s	Set to 1 if block i is assigned to resource j in the timeslot s; 0 otherwise
y_{ij}	Set to 1 if block i is assigned to resource j; 0 otherwise
z_{is}	Set to 1 if block i is assigned in the timeslot s; 0 otherwise
T_i	Ending timeslot for block $i \in \upsilon$, $T_i \in \mathbb{N}$
t_i	Starting timeslot for the block $i \in \upsilon$, $t_i \in \mathbb{N}$
T	Ending timeslot for the whole workflow, $T \in \mathbb{N}$
t	Starting timeslot for the whole workflow, $t \in \mathbb{N}$

The objective function for the problem is thus expressed as the minimization of the difference between ending timeslot and the starting timeslot of the workflow; in other words, the goal is that of minimizing the makespan of the workflow execution (it is worth to mention that minimizing the makespan is equivalent to maximize the packing efficiency [4]):

$$\min\ (T - t)$$

The constraints of the programming model are defined to properly represent the requirements of the resource scheduling problem; particular attention is paid to provide a linear formulation. Thus, the resulting overall model is a constrained Integer Linear Program (ILP).

To properly retrieve the makespan of the whole workflow, T is the maximum of the ending times of the blocks and t is the minimum of the starting time:

$$T_i \leq T$$
$$\mathrm{t} \leq t_i$$

In the solution, the dependencies of the blocks must be preserved, so the starting time of each block should be greater than the ending time of the preceding blocks in the workflow:

$$T_i < t_k \qquad \forall i \in \upsilon,\ \ \forall k \in \mathcal{N}_i^+$$
$$\mathrm{s} \cdot \mathrm{z}_{is} \leq T_i \qquad \forall i \in \upsilon,\ \ \forall s \in \mathcal{T}$$
$$\mathrm{s} \cdot \mathrm{z}_{is} \geq t_i \qquad \forall i \in \upsilon,\ \ \forall s \in \mathcal{T}$$

When a block is assigned to a certain resource, it must run for all the required time on that specific resource:

$$\sum_{p=s}^{s+w_i} \mathrm{x}_{ij}^p = w_i \cdot y_{ij} \qquad \forall i \in \upsilon,\ \ \forall j \in \mathcal{R},\ \ \forall s \in \mathcal{T}$$

$$\sum_{j\in\mathcal{R}} y_{ij} = r_i \quad \forall i \in \upsilon$$

When a block is assigned to a specific timeslot, it should have access to all the computational resources it needs:

$$\sum_{k\in\mathcal{R}} \mathrm{x}_{ik}^{s} = r_i \cdot z_{is} \qquad \forall i \in \upsilon, \quad \forall s \in \mathcal{T}$$
$$\sum_{s\in\mathcal{T}} z_{is} = w_i \quad \forall i \in \upsilon$$

In a specific timeslot, for a certain resource, just one block can be scheduled, thus leading to the following constraint:

$$\sum_{i\in\upsilon} \mathrm{x}_{ij}^{s} \le 1 \qquad \forall j \in \mathcal{R}, \quad \forall s \in \mathcal{T}$$

In this formulation it is important that some of the variables will assume a constant value, thus actually not being proper optimization variables, but acting as constant parameters: compute resources that do not satisfy the requirements for a certain block allocation are cut out from the possible solution; in the same way if a certain timeslot is not available for a certain resource it is not considered. This leads to setting variables as reported in Table 2:

Table 2 - Settings of model variables as constants

Variable	Description
$x_{ij}^{s} = 0$	When the assignment of block i to timeslot s and to resource j is not possible
$y_{ij} = 0$	For all resources j that are not compatible with the requirements of block i
$z_{is} = 0$	When no resource is available at timeslot s for block i

2.3 Workflow-aware Advance Resource Planner (WARP)

The Workflow-aware Advanced Resource Planner (WARP) is the orchestration component that oversees executing the solver function for the optimization problem stated in Section 2.2.1. As such, the module has to:

1. Keep an ordered list of the workflows, from which to select the current one to optimize the resource allocation; for the selected list, the minimal graph representation should be retrieved from StreamFlow.
2. Retrieve the information from the batch scheduler concerning the current scheduling of the jobs (see Figure 3, coloured squares) which define the starting condition for the optimization process.
3. Solving the instance of the allocation problem and through the batch scheduler creating the proper resource reservations.

To accomplish with these three steps, a modular architecture has been devised. The architecture foresees a *frontend* component which exposes a command line interface (CLI) through which the user can interact with the system, as well as the other components (e.g., StreamFlow) that can perform specific operations. We expect to have a frontend instance running separately for each entity (user, orchestration component) that needs to perform some action (e.g., check the status of a workflow, perform a reservation, etc.). A second component is represented by the backend, which asynchronously interact with the batch scheduler to retrieve the information related to the current job schedule, as well as to perform the reservations. Given the limited visibility of the internal information kept by the batch scheduler, a plug-in has been devised as the best way to expose the required information, as well as to provide the right way of interacting with the batch scheduler. Indeed, it is important (due to the asynchronicity in the process, which implies to introduce a latency between the point in time in which status of the jobs' schedule is collected and the one in which reservation is made) to guarantee that the reservation is requested for a point in time in the future that is not yet occupied by any scheduled job in the meantime.

To properly keep the information concerning the reservations made and the status of the workflows that are executing, as well as the information regarding the other workflows in the ordered list, a *permanent data store* is created. This data store is implemented as a SQL database, which can manage concurrent read/written operations issued by the WARP components. Structure of the WARP module is depicted in Figure 4 where, in a first instance, we see the data store being managed directly by the backend component.

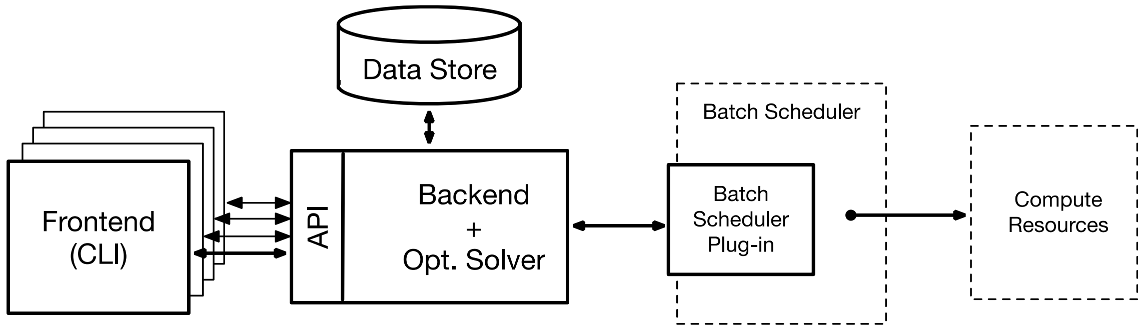

Figure 4 - High level architecture of the WARP module, along with the batch scheduler plugin.

The data store is organized into a set of tables that persistently keep track of the submitted workflows, their structure, execution state and reservations made. Considering the structure reported in Figure 4, the interaction between the components is as follows:

1. Users submit a new workflow (described as a CWL file coupled with a StreamFlow file) through the WARP CLI frontend. As such, the frontend, launches a StreamFlow instance which will be responsible for running that specific workflow. The WARP frontend also keeps track (in the data store) of the process ID associated to the specific StreamFlow instance.
2. From the CWL and StreamFlow files, the minimal graph representation for the workflow's steps is derived and written in the data store. This step can be accomplished by StreamFlow through a specific command exposed by the WARP frontend.
3. Once the graph representation is available, the WARP backend can order the workflows as described in Section 2.2.1 and get the one with the highest priority with regards to the used ordering parameter(s) –e.g., workflows' arrivals time, resources' requirements of the jobs and priority scores.
4. Given the selected graph, the reservations can be made. To this end, the WARP backend contacts the batch scheduler plugin to retrieve the information concerning the current schedule and determine the proper point in time starting from which to make the reservation.
5. Once the reservation is available, the WARP backend writes back the associated information (i.e., reservation name to be use for job submission) on the data store. Then, periodically the StreamFlow instance that was associated to the workflow will poll the backend to see when the resources will be available for running jobs:
    - StreamFlow takes care of executing the workflow's steps once the reserved resources become available.

This execution flow is exemplified in Figure 5.

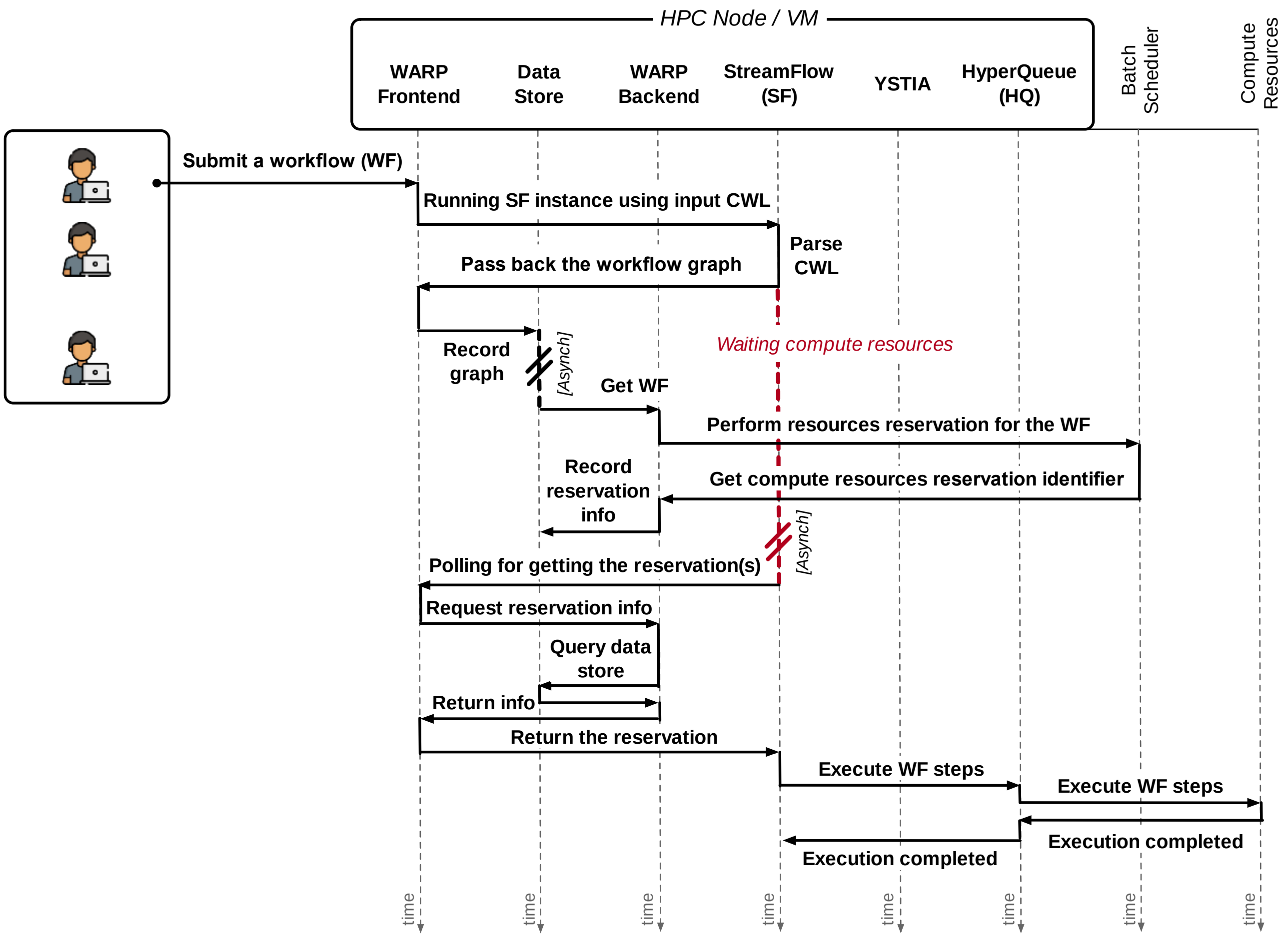

Figure 5 - Interaction flow among different components of the ACROSS orchestrator.

3 Resources Usage Optimization

The optimization of resource usage is achieved by controlling the reserved resources with a fine granularity. Generally, HPC resources can be acquired by submitted jobs with the granularity of the entire single node, making them in some cases over-provisioned. For instance, heterogeneous nodes where both CPU cores and GPUs are available are partially used if the running jobs use only the CPU or GPU part. To overcome this limitation, in ACROSS, the orchestrator relies on the HyperQueue (HQ) [6] tool to provide more control over the reserved resources. Through this tool it will be possible to better optimize the resource usage and fulfil the co-location requirement.

On the other hand, cloud resources should be properly managed. The acquisition process of cloud resources is quite different from that of HPC resources. Generally, cloud resources (in this case we refer to them as virtualized resources) are requested on-demand, and they must be explicitly released once the execution of the jobs complete. Also, the execution of jobs on these virtualised resources may require the installation of the supporting execution environment (deploying); to this purpose, tools for automating these processes are used. In ACROSS we envisioned the orchestrator in such a way it will be able to target the use of both HPC and cloud resources when needed. To accomplish with this goal, YSTIA modules are integrated in the high-level orchestrator architecture. As such, workflows that require the access to cloud virtual instances will be managed too. FastML Engine (FMLE) tool has been designed to take advantage of the HPC resources to perform the training of large machine learning and deep learning models; however, FMLE is flexible enough to eventually target cloud resources as the main compute horsepower, when training performance are not the primary concern. As such, one case where we see the access to cloud resources is that of FMLE. Indeed, ACROSS aims to also include the management of Spiking Neural Networks (SNNs) under the umbrella of FMLE, leveraging the capability of YSTIA to allocate/deallocate cloud resources, deploying/undeploying execution environments.

3.1 Fine-grain allocation of HPC resources

HyperQueue (HQ) [6] is a tool developed by IT4Innovations to simplify the execution of large workflows on supercomputers. As such, the tool receives the workflow to execute and automatically requests appropriate resources from the batch scheduler; once resources become available, it schedules the execution of tasks on those resources. In this context, it is worth to mention that, on one side, the unit of computation that can be scheduled is a *task* (i.e., a single execution of a program, for instance), in contrast to what is managed by the batch schedulers —i.e., a job, that appears to HQ as a composition (a graph) of many tasks. Jobs are still managed by HQ, but for the purpose of acquiring compute resources. On the other hand, it is worth to mention, that HQ can allocate fractions of the whole acquired resources to the tasks, having the notion/visibility of the resources at a finer grain with respect to batch schedulers.

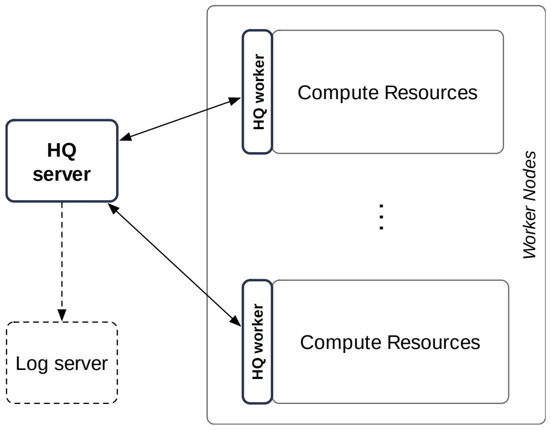

Figure 6 - HyperQueue (HQ) architecture

This translates in being able to assign from a single core to an entire CPU socket to a task, being able to specify the policy of pinning tasks to CPU cores (by default, HQ enforces the scheduling of concurrent tasks on those workers that has enough resources available to fulfill the request, but leaving the system scheduler to decide how to move threads among the CPU cores/sockets), and how much memory each task should use. It has also the ability to scale from a single node to several nodes.

To accomplish with these capabilities, HQ leverages a master-workers model (see Figure 6), meaning that there is a master entity (HQ server) which is responsible for interacting with users (or external components of the orchestrator), receiving the workflow and scheduling the tasks on the workers. The scheduling policy uses a work-stealing approach which is based on Tako [7] and RSDS (Rust Dask Scheduler) [8] previously designed and implemented by IT4Innovations. Workers can (directly) exchange data with the HQ server bypassing the batch scheduler's queues (eventually, workers can communicate each other), resulting in a fast interaction between the acquired resources and the application using them. Eventually, HQ can be configured in such a way, the output of the execution of the task-graph is streamed out to an external server where a log file is generated.

From the perspective of ACROSS, all these capabilities are useful in pursuing the goal of optimizing the usage of compute resources in the managed systems. Indeed, being able to control the allocation of tasks with the granularity of the CPU core, as well as being able to target the execution on specific resources (e.g., a GPU or a FPGA accelerator) makes HQ the right complement to the WARP and Streamflow modules. In this context, it will be the WARP module that acquires the resources and makes them available to the HQ server/workers. HQ will be used by StreamFlow for submitting the tasks to execute on the resources reserved by the WARP module.

At the time of writing this deliverable, the HQ development team is working on extending its capability even more than what described above. Specifically, it is targeting to introduce the support for tasks requiring more than one worker to execute (e.g., MPI tasks). Also, this feature will be exploited to better support the execution of ACROSS pilot workflows.

3.2 Cloud resources management

FastML Engine (FMLE) [11] is a tool designed by ATOS aimed at providing useful features for data scientists to manage the steps related to modelling and training machine learning (ML) and deep learning (DL) models. To this end, it manages the underlying operations needed to train the models (eventually using a distributed approach) focusing on using HPC resources, managing the hyperparameter optimization process, as well as managing different development frameworks (TensorFlow, Keras, Scikit-Learn, PyTorch, etc.). In ACROSS, we foresee the usage of FMLE to support the modeling and training phases of ML/DL models that are defined in the pilots; furthermore, we see FMLE as a building block for implementing a training toolchain supporting the integration of dedicated tools to convert models into SNNs, supporting neuromorphic accelerators for the inference stage. As such, our intention is that of allowing FMLE to access the needed compute resources for performing the modeling/training operations, which can be both those on the HPC cluster and the cloud partition. In this latter case (for instance, when training performance are not a major concern, a cluster of virtual machines can be created and used for that purpose), it is needed to have a tool supporting the phases of requesting virtual resources, deploying the necessary software frameworks, undeploying software and relinquishing virtual resources. FMLE has been designed to integrate with the Ystia toolset, which provides all the necessary machinery for accomplishing those needs.

Ystia is a toolset developed by ATOS to manage all the phases concerned with the allocation and usage of compute, storage, and network resources on the cloud. As such, Ystia comes with two main components that provides basically an interface (UI and CLI) and the actual engine performing all the necessary steps to acquire, configure and then release resources (i.e., virtual machines). These two components are namely Alien4Cloud and Yorc respectively.

Alien4Cloud (A4C) [12] is the front-end and contains an extensible catalogue of TOSCA (i.e., an interoperable standard for describing cloud applications and the way they should be deployed on a set of resources) components, an environment to design applications from this catalogue, and associated sequence of steps for the deployment (i.e., an execution plan). It relies on an execution engine (by default Yorc – i.e., Ystia orchestrator [13]) to manage application lifecycles and to run execution plan on the infrastructures supported by the engine. This engine is TOSCA compatible and supports the application lifecycle management over hybrid infrastructures (i.e., HPC clusters, Kubernetes, OpenStack, public clouds). It provides an implementation of operations allowing the creation/deletion of virtualized (infrastructure) resources (e.g., compute instances, block-devices, etc.) on demand. It also allows the installation of applications and to run execution plans on these infrastructures. By design, it can be extended by means of dedicated plugins, for instance to support additional infrastructures. As mentioned above, Ystia relies on a catalogue of software components; there is a publicly accessible catalogue named Ystia Forge that provides TOSCA components and application templates that can be imported in the A4C component. For example, it provides the implementation of a component that allows the installation of docker on a certain host and another TOSCA component aimed at supporting the running of a docker container.

Interestingly, the access to cloud resources within the ACROSS project, is not limited to the perimeter of providing the compute, storage, and network resources to support FMLE, but we foresee the possibility to support directly the execution of applications (workflows); indeed, every time it is necessary to perform a visualization, pre- or, post-processing steps that does not strictly require execution on HPC cluster's resources, it can be done on virtual machines whose configuration can be tuned time by time, and managed via the Ystia tools.

4 Conclusions

One of the goals of the ACROSS project is to design and integrate a set of hardware and software technologies to improve the execution of complex workflows as provided in WP5, WP6 and WP7. To this end, in addition to the improvements –i.e., faster execution of applications– achieved by using faster hardware (CPUs) and accelerators (e.g., GPUs, FPGAs, etc.), major gains can be achieved by developing an innovative orchestration system. An important aspect in this context is the optimal management of the computing resources, as they are the primary target (on one side) of the application level, and (on the other side) of the allocation and scheduling policies.

Therefore, this deliverable focuses on the analysis and definition of allocation and scheduling strategies that will be implemented in the ACROSS orchestration system to enable such optimal management of resources. To this end, we have begun by considering the outcomes of other deliverables where we have collected and analysed the system requirements that underlie the high-level orchestrator architecture, and then we focused on the specific components that deal with the management of the life cycle of (computational) resources. In this sense, the role of the WARP module, which we developed as one of the components of the orchestration architecture, was analysed in detail.

In view of this, the deliverable examines what are the main constraints (found) in properly managing resources having the visibility on the workflows to be executed. That gave us the opportunity to envision a different strategy to acquire the required resources, with the objective of ensuring a deterministic execution, and thus reducing the time generally wasted by jobs in the queues of batch schedulers. The deliverable describes (after properly specifying the above constraints) the optimization model (i.e., an ILP model) in which our proposed resource management (i.e., resource allocation) strategy is embedded. The optimal use of computational resources is not limited to the orchestrator's ability to properly acquire these resources, but also to the way in which the resources can be controlled and allocated to the various tasks to be executed. Therefore, in Chapter 3, we described two additional components of the architecture that allow HPC resources to be managed and controlled in a fine-grained manner (i.e., allocating resources at the granularity of CPU cores), as well as cloud resources.

To summarize the contribution of this deliverable, we have found that current (State-of-the-Art) batch schedulers, even though they can maximize resource utilization and maintain fairness among users, do not have the right visibility of the workflows that need to be executed. As workflows become larger and more complex, this shortcoming must be addressed by implementing appropriate policies, possibly in addition to existing tools. To address this requirement, in ACROSS we have focused on leveraging the specific features of batch schedulers, to develop on top of the batch scheduler the logic that governs the acquisition of resources and their allocation to tasks.

References

[1] Fan, Yuping, et al. "Scheduling beyond CPUs for HPC." *Proceedings of the 28th International Symposium on High-Performance Parallel and Distributed Computing*. 2019.

[2] Fan, Yuping, and Zhiling Lan. "Exploiting multi-resource scheduling for HPC." *SC Poster* (2019).

[3] Sanjay Kadam et al., Bio-inspired workflow scheduling on hpc platforms. *Tehniˇcki glasnik*, 15(1):60–68, 2021

[4] Robert Grandl, Ganesh Ananthanarayanan, Srikanth Kandula, Sriram Rao, and Aditya Akella, "Multi-resource packing for cluster schedulers", *ACM SIGCOMM Computer Communication Review*, 44(4):455–466, 2014

[5] Fan, Yuping, and Zhiling Lan. "DRAS-CQSim: A reinforcement learning based framework for HPC cluster scheduling." *Software Impacts* 8 (2021): 100077.

[6] https://github.com/It4innovations/hyperqueue

[7] https://github.com/spirali/tako/

[8] https://github.com/It4innovations/rsds

[9] ACROSS deliverable *D4.1* – *System Requirements Analysis for Orchestrator Design* https://www.acrossproject.eu/deliverables/

[10] ACROSS deliverable *D2.2* – *Description of key technologies and platform design* https://www.acrossproject.eu/deliverables/

[11] FastML Engine (FMLE) – Part of Codex AI Suite (https://atos.net/fr/solutions/codex-ai-suite)

[12] Alien4Cloud (https://alien4cloud.github.io/)

[13] Yorc – Ystia orchestrator (https://ystia.github.io/)

[14] StreamFlow – https://streamflow.di.unito.it