Advanced Resource Allocation in the Context of Heterogeneous Workflows Management

Advanced Resource Allocation in the Context of Heterogeneous Workflows Management

Francesco Lubrano
francesco.lubrano@linksfoundation.com
LINKS Foundation
Torino, ITALY

Chiara Vercellino*
LINKS Foundation
Torino, ITALY

Giacomo Vitali†
LINKS Foundation
Torino, ITALY

Paolo Viviani
LINKS Foundation
Torino, ITALY

Alberto Scionti
LINKS Foundation
Torino, ITALY

Olivier Terzo
LINKS Foundation
Torino, ITALY

ABSTRACT

In High-Performance Computing (HPC), workflows are utilized to define and manage a set of interdependent computations which allow the users to extract insights from (scientific) numerical simulations or data analytics. HPC platforms can perform extreme-scale simulations, combining Artificial Intelligence (AI) training and inference and data analytics (we refer to heterogeneous workflows), by providing tools and computing resources which serve a variety of use-cases spanning very diverse application domains (*e.g.*, weather forecasting, quantum mechanics, etc.). Executing such workflows at scale requires to handle dependencies, job submission automation, I/O mechanisms. Despite State-of-the-Art batch schedulers can be configured and integrated with tools accomplishing this automation, a number of cases where resource allocation can lead to inefficiencies still exist. In this paper, to overcome these limitations, we present the WARP (Workflow-aware Advanced Resource Planner), a tool that integrates with workflow management tools and batch schedulers, to reserve in advance resources for an optimal execution of jobs, based on their duration, dependencies and machine load. WARP has been designed to minimize the overall workflow execution, without violating the priority policies for cluster users imposed by the system administrators.

CCS CONCEPTS

• **Software and its engineering** → **Software as a service orchestration system**; **System description languages**; • **Computing methodologies** → *Machine learning*; **Planning and scheduling**.

KEYWORDS

High Performance Computing; Queues; Batch scheduler; Workflows; Optimization; Cluster load distributions.

*Also with Politecnico of Torino.
†Also with Politecnico of Torino.

WiDE '24, April 22, 2024, Athens, GR

ACM ISBN 979-8-4007-0546-5/24/04
https://doi.org/10.1145/3642978.3652835

ACM Reference Format:
Francesco Lubrano, Chiara Vercellino, Giacomo Vitali, Paolo Viviani, Alberto Scionti, and Olivier Terzo. 2024. Advanced Resource Allocation in the Context of Heterogeneous Workflows Management. In *WiDE'24: 2nd Workshop on Workflows in Distributed Environments, April 22, 2024, Athens, GR.*. ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3642978.3652835

1 INTRODUCTION

High-Performance Computing (HPC) systems are a crucial asset in many domains, such as scientific research, engineering, and data analytics. Thanks to the huge processing power, HPC platforms (aka supercomputers or large clusters) can perform among others, extreme-scale simulations, AI model training and inference, and data analytics on large datasets, which are now the fundamental steps that allow industries to simulate innovative designs, and researchers to get new insights in a variety of domains *(e.g.*, weather forecasting, astrophysics simulations, quantum mechanics, etc.).

The flexibility of serving such a diverse use-cases catalyzed the interest on efficiently managing *heterogeneous* workflows, *i.e.*, a composition of (dependent) computations (referred to as workflow steps) belonging to different domains (*e.g.*, numerical simulations, AI model training, in-situ processing and visualization, etc.) [14, 20, 23], which are characterized by different requirements. However, batch schedulers implement priority queues and policies that maximize resource utilization at the expense of single jobs or workflows execution time, thus making the case for a better allocation of HPC resources. When working with large-scale workflows, which involve a series of interconnected computations that need to be executed in a specific sequence, it becomes essential to introduce mechanisms that add more execution determinism, while ensuring the best usage of resources.

For this reason, one of the main objectives of the advanced orchestration solution developed within the ACROSS project[1] is to enable the possibility of reserving *on-demand* HPC resources for a given workflow. Workflows are made of many steps (jobs), *i.e.*, computational blocks, that can be executed in parallel or have many input-output dependencies. These dependencies are provided in the workflow description files, which follow, in our case, the Common Workflow Language (CWL) [2] specification; the ACROSS orchestration solution leverages such information to optimize workflow execution. The optimization lies in determining the best resource

[1]https://www.acrossproject.eu/
[2]https://www.commonwl.org/

allocation, aiming at minimizing the overall execution time, and maximizing resource utilization (or reducing the cost of resource utilization) while fulfilling the job constraints. These constraints may include job dependencies, deadlines, memory requirements, and the availability of specific hardware configurations. Additionally, the allocation strategy should consider the dynamic nature of HPC environments, where resource demands change over time due to varying workloads and user/queue priorities. To automate the (on-demand) reservation mechanism, and to optimize the reservation shapes and their placement over time, the WARP has been developed and integrated into the whole orchestration stack. The reservation mechanism allows to obtain two main advantages: *i)* optimize the allocation and the usage of resources, through deterministic and optimized scheduling, and *ii)* save user time, effort, and money thanks to the dynamic definition of reservation shapes and automatic management of workflow step dependencies. Although required to achieve computing efficiency, at this stage of development, the management of heterogeneous computing resources is left out the optimization process, leaving this task to a fine-grain resource scheduler. Context related to this work is provided in the section 2 and section 3, then this paper presents the details of the WARP in section 4 and reports simulation and demonstration results in sections 5 and 6. Section 7 concludes the paper.

2 RELATED WORKS

Workflows are a powerful mechanism to abstract the description and execution of (scientific) applications on a given infrastructure. Their management requires the use of dedicated workflow management systems (WMS) to properly handle the execution of each step (job) on a specific set of compute resources.

The landscape of WMS is variegated and spans from high-level tools aimed at supporting workflow designs (often leveraging domain-specific knowledge) to low-level tools targeting the efficient scheduling of jobs on computing resources [4, 5, 7, 15]. WMS designed to fulfill the requirements of the scientific community in one specific domain, offering robustness and automation over data transfers, emerged since the initial diffusion of web services and grid computing technologies [1, 9, 18]. Other frameworks provide a lower-level interface which allows expressing job (in some cases we can refer to tasks within a job) dependencies directly at the programming level (*e.g.*, COMPSs [16]). Tools like Apache Airflow [11] and Snakemake [13] provide the users with a domain-specific language (DSL), to programmatically author and manage workflows. StreamFlow [6] has been proposed to close the gap between the expressiveness of the language used to describe the workflows, and the flexibility and modularity to target different types of infrastructures and computing resources.

Resource allocation is per se a vast research topic; over the years, the scientific community released algorithms and mechanisms to make (mostly compute) resource allocation more robust and effective, targeting different types of infrastructures (HPC, Cloud, distributed environments). In [12], the authors surveyed the algorithms and tools used to allocate resources in HPC-distributed systems. Netti et al. [17] proposed three heterogeneity-aware resource allocation algorithms, optimizing allocation and use of specialized hardware and heterogeneous resources, and demonstrating improvements in terms of job response times and system throughput. Others attempted to provide algorithms focusing on other aspects like reliability [19] and energy efficiency [8].

With the continuous and growing shift towards workflows pairing numerical simulations with data analytics even backed by machine learning and deep learning tasks (heterogeneous workflows), HPC centers faced the limitations of traditional resource and job management software (*e.g.*, SLURM, PBS, etc.). In [2, 3] a hierarchical flexible system (Flux) handling complex workflows with efficient use of allocated compute resources was proposed. Similarly, Balsam [21] has been proposed to ease the execution of heterogeneous workflows on first-class supercomputers. These systems leverage on low-level resource managers (*e.g.*, SLURM) to acquire resources (through job submission) and allocating more powerful, hierarchical, schedulers on top to distribute the workload. Our work mainly differentiates by a different use of the low-level resource manager (SLURM), to meet deterministic execution. On the other hand, a hierarchical integration of fine-grain schedulers is still possible with our solution.

3 SUPPORTING HETEROGENEOUS WORKFLOWS EXECUTION

The complexity of modern workflows follows side-by-side the evolution of computing infrastructures. In this regard, (HPC) infrastructures, spanning from small research clusters to supercomputers to (public) Clouds, became very heterogeneous, including different flavours of hardware accelerators (*e.g.*, GPUs, TPUs, AI-specific architectures, etc.). In the same way, workflows may leverage this large heterogeneity to break the boundaries between application domains (*e.g.*, numerical simulations, machine learning –ML and deep learning –DL training and inference, etc.). Complex heterogeneous workflows allow both exploiting diversity in compute capabilities and improving user productivity. Whilst all this opens the door to many innovations, it comes also with challenges to be addressed. Among the others, properly managing the allocation of computing (also networking, storage) resources becomes of paramount importance. Indeed, a WMS is in charge of capturing this complexity and promptly responding to resource allocation requests.

3.1 The ACROSS Project

The ACROSS project was funded in 2021 under the H2020 and EuroHPC-JU frameworks; it targeted to co-design and implement an orchestration software stack aimed at enabling workflows to leverage current petascale and pre-exascale machines. By specifically targeting *heterogeneous* workflows whose steps encompass numerical simulations, ML/DL training and inferences, and HPDA (High Performance Data Analytics) procedures, this orchestration stack has been architected to *i*) provide a unified mechanism to describe heterogeneous steps along with their corresponding computing resources, *ii*) provide a more deterministic execution, and *iii*) exploit compute resources at lower granularity than that of a single node. To this end, many software components have been integrated together. A workflow management system (*i.e.*, StreamFlow), has been enhanced through the integration of a module aimed at dynamically allocating resources to the workflows' steps (WARP

–see Section 4). This latter is paired to a batch scheduler plugin providing more control over the allocation of resources w.r.t. the exposed standard queuing system. Finally, a fine-grain resource scheduler is added, allowing for further resource splitting with a very fine granularity, generally below the single node granularity (*e.g.*, co-scheduling tasks using GPU resources of a node and leaving almost all the CPU resources of the node for other tasks). One of the main challenges related to this architecture lies in the creation of effective allocation strategies. The following section details the two main approaches pursued during the project's lifetime.

4 ADVANCED RESOURCE ALLOCATION STRATEGIES

Workflow-aware Advanced Resource Planner (WARP) is the tool developed to manage and determine the shape and characteristics of the reservations, *i.e.*, defining the number of compute nodes to be allocated and specifying a starting time and a time duration. It is written in Rust and leverages Common Workflow Language (CWL) standard. WARP can be divided into several logical parts which are interconnected and interdependent.

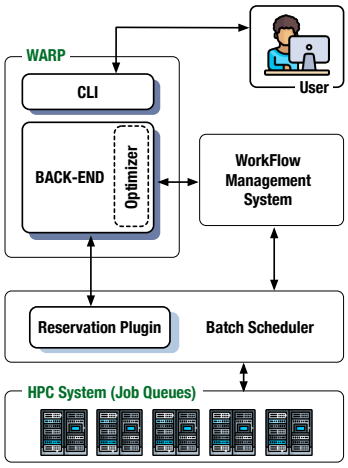

Figure 1: WARP internal structure and interconnections

Figure 1 depicts the internal structure of the WARP and its interconnection with external tools. WARP is made of a Command Line Interface (CLI), which allows the user to submit, inspect, and manage workflows, and a back-end part, which implements the main logic. One or more users can submit workflows to the WARP. The internal logic, leveraging the workflow description and configuration files provided by the user, identifies the steps defining the workflow and their dependencies. The output of this parsing is a representation of the workflow as a graph. Every 10 minutes, an optimization cycle is triggered. All the steps that are waiting for a reservation, along with the current load on the batch scheduler, the jobs in the queues, the active reservations, the characteristics of the cluster are provided to the optimizer. At the end of the cycle, optimizer module outputs a number of reservations to be allocated on the cluster. WARP can submit reservation through the interaction with a specific batch scheduler plugin. Indeed, we developed a plugin for the SLURM scheduler, that implements a new SLURM command, *i.e.*, *sresv*, allowing to reserve on-demand resources on the cluster. Moreover, this plugin allows to perform reservations without administrative privileges. Scheduler plugins are a convenient mechanism to add specific functionalities to batch scheduler and do not interfere drastically with internal scheduling policies, limiting issues or mistrust when dealing with an HPC production environment. At the end of the reservation cycle, if the reservation has been allocated correctly, each step is associated with a reservation. Through a polling mechanism, the workflow manager (used as a workflow execution engine) can get the array of steps ready to be submitted to the batch scheduler.

4.1 Job Queue Waiting Time Predictor

One potential path to enhance the user experience within the workflow is to forecast the behavior of cluster queues. In an ideal scenario, where the status of queues or cluster resources can be accurately anticipated, such as predicting the queuing time of HPC jobs, one could pinpoint optimal submission times for workflow steps. This would satisfy step dependencies and minimize the overall makespan - the total duration from the initiation of the first step to the completion of the final dependent steps.

This aspiration prompted us to explore the utilization of machine learning (ML) techniques to refine workflow execution, specifically in predicting the time jobs spend awaiting execution in batch scheduler queues. In our prior research, we introduced an automated procedure that combines Unsupervised Learning (UL) and Supervised Learning (SL) techniques to anticipate queue waiting times in an HPC environment [22]. The UL phase involved preprocessing, employing advanced methods to uncover patterns in job features and identify crucial elements for prediction. Subsequently, the SL phase facilitated actual predictions of the target feature, namely queue waiting time.

Within our investigation of various SL models, we assessed different classification and regression models to pinpoint potential candidates using the analyzed dataset.

Moreover, we underscored pivotal considerations when addressing analogous case studies. Our examination of different dataset splits highlighted the significance of temporal components influencing data distributions. Consequently, we proposed an uncertainty quantification approach to evaluate prediction reliability and discern correlations with error distributions.

After our initial foray into ML techniques for forecasting queue waiting times in an HPC cluster, we acknowledged that the outcomes fell short of our expectations. Consequently, we shifted our focus towards utilizing data-fitted models to delve deeper into inter-arrival patterns, resource demands, and walltime distributions. This strategic pivot empowered us to refine job submission strategies by leveraging insights gleaned from historical cluster data. As a result, we transitioned away from merely predicting queue times and towards employing models that comprehensively capture job characteristics and cluster dynamics. This facilitated the creation of a simulated test environment conducive to evaluating WARP reservation-based solutions.

4.2 On-Demand Reservations

Given the challenges associated with predicting the behavior of HPC queues, we determined that a more effective approach for orchestration within the ACROSS project was to utilize on-demand reservations. In this second approach, the WARP receives information from the back-end regarding the cluster status. By analyzing the jobs in the queue and those currently running, it can ascertain the number of computational resources already allocated over time. Additionally, the WARP back-end provides details on the workflows, represented as Directed Acyclic Graphs (DAGs), awaiting scheduling. Specifically, the optimizer is supplied with two key inputs. First, it receives the cluster occupancy status, which reflects the number of nodes already allocated for other HPC jobs (either pending or running) within each timeslot. Notice that to address this scheduling optimization, time is discretized into timeslots of 1 minute. Secondly, the optimizer receives the description of the workflow, encompassing dependencies, the required number of nodes for each HPC job (referred to as computational *blocks*) within the workflow, and their corresponding expected execution time (referred to as *walltime*).

The elements of the optimization problem tackled by the optimizer are as follows:

- $\mathcal{G}(\mathcal{V}, \mathcal{E})$, the DAG representing the workflow to be allocated; $\mathcal{V}$ is the set of blocks, each characterized by the number of HPC nodes required r_i, and the execution walltime w_i (expressed in minutes); the set of directed edges $\mathcal{E}$ defines the dependencies among computational blocks and implies the allocation order; $N_i^+ = \{j \in \mathcal{V} : i \rightarrow j \in \mathcal{E}\}$ describes, for each $i \in \mathcal{V}$, the execution blocks that must be allocated after it.
- N_T, the last timeslot available for the scheduling.
- $\mathcal{T}$, the set of the timeslots, from 0 to N_T.
- N_c, the total number of nodes in the cluster, and $n(s)$ is the number of nodes already allocated in timeslot $s \in \mathcal{T}$.

The optimization variables of the scheduling problem are:

- z_{is} := binary variable with value 1 if block i is assigned to timeslot s;
- T_i := ending timeslot for block $i \in \mathcal{V}$;
- t_i := starting timeslot for block $i \in \mathcal{V}$;
- T := ending timeslot of the whole workflow;
- t := starting timeslot of the whole workflow.

The optimization problem's objective function is defined by the total makespan of the workflow execution, with the goal of minimizing it. Notably, minimizing the makespan is synonymous with maximizing packing efficiency [10].

Hence, the overarching optimization problem is framed within the Integer Linear Program (ILP) outlined by (1a)-(1l). Constraint (1b) ensures that each computational block $i \in \mathcal{V}$ lasts exactly its walltime w_i; constraints (1c) and (1d), in conjunction with the objective function (1a), enable the optimizer to accurately determine the makespan of the workflow: T is the maximum of the ending times of the blocks and t is the minimum of the starting time.
Preserving the dependencies of the blocks is crucial in the scheduling solution. Consequently, the starting time of each block must exceed the ending time of the preceding blocks in the workflow, as expressed by constraint (1e). The constraints (1f) and (1g) establish the connection between the binary variables z_{is} with integer variables T_i and t_i. Constraint (1h) ensures that the allocation of resources within a timeslot does not exceed the available resources. Additionally, constraint (1i) guarantees that for each block in the DAG, there are precisely w_i binary variables with a value of 1. Lastly, constraints (1j), (1k) and (1l) delineate the domains of the optimization variables.

$$\min \quad T - t \tag{1a}$$

$$\text{s.t.} \quad T_i - t_i = w_i - 1 \qquad \forall i \in \mathcal{V}, \tag{1b}$$

$$t \leq t_i \qquad \forall i \in \mathcal{V}, \tag{1c}$$

$$T_i \leq T \qquad \forall i \in \mathcal{V}, \tag{1d}$$

$$T_i + 1 \leq t_k \qquad \forall i \in \mathcal{V}, \forall k \in \mathcal{N}_i^+, \tag{1e}$$

$$s \cdot z_{is} \leq T_i \qquad \forall i \in \mathcal{V}, \forall s \in \mathcal{T}, \tag{1f}$$

$$(N_T - s) z_{is} \leq N_T - t_i \qquad \forall i \in \mathcal{V}, \forall s \in \mathcal{T}, \tag{1g}$$

$$\sum_{i \in \mathcal{V}} z_{is} \cdot r_i \leq N_c - n(s) \qquad \forall s \in \mathcal{T}, \tag{1h}$$

$$\sum_{s \in \mathcal{T}} z_{is} = w_i \qquad \forall i \in \mathcal{V}, \tag{1i}$$

$$z_{is} \in \{0, 1\} \qquad \forall i \in \mathcal{V}, \forall s \in \mathcal{T}, \tag{1j}$$

$$t_i, T_i \in \{0, \ldots, N_T\} \qquad \forall i \in \mathcal{V}, \tag{1k}$$

$$t, T \in \{0, \ldots, N_T\} \tag{1l}$$

In our current setup, we utilize the GLPK (GNU Linear Programming Kit). We impose a time constraint of 10 seconds for finding a solution, which proves to be ample for our specific case-study as we consistently achieve optimal solutions within 1-2 seconds.

However, should we scale up to a cluster with a higher number of compute nodes, the computational complexity involved in solving the ILP optimally would exceed the time limits. In such scenarios, transitioning from an exact solver to a heuristic approach becomes necessary to maintain feasibility, albeit at the expense of optimality. It's worth noting that while our ILP formulation currently assumes homogeneous resources, adapting it to accommodate heterogeneous resources is straightforward and would not alter the programming model's structure.

The solution of the ILP informs on-demand reservations that are made possible by integrating a dedicated batch scheduler plugin, which takes care of creating the reservations in terms of duration and number of resources to allocate. This makes possible to decouple the reservation requests which a restrained to the user space from their actual implementation which lies at the system level. Looking at the specific case of the SLURM batch scheduler (which is widely adopted by many HPC centers), the approach to achieve this is as follows: *i*) the information describing the reservation is passed through the WARP to the batch scheduler as it was a job to be queued; *ii*) the request is enriched with flags forcing the job to never enter the targeted queue; *iii*) the system checks if the request (job) can be fulfilled, *i.e.*, there are enough resources in the near future; and *iv*) if the request can be fulfilled, the job is substituted with a corresponding reservation on the system. This approach avoids negatively impact on the internal scheduling policies already in place on the batch scheduler. Notably, although we targeted the specific case of SLURM, other batch schedulers and WMS can be

tweaked to provide the same functionality, although the complexity of this operation depends on their specific internal architecture.

5 SIMULATION ENVIRONMENT

To develop, test, and validate the WARP, we created an isolated testbed infrastructure from a partition of a real HPC platform. The total amount of vCPUs available for this simulation environment have been divided into defining: *i)* 1 virtual node, which exports the whole storage quota (3.0TB) through an NFS service; *ii)* 2 virtual nodes running SLURM services, WARP, and StreamFlow (the WMS selected for these tests and integrated with WARP); *iii)* 14 virtual nodes managed by the batch scheduler. Each node is connected to a common virtual network. The SLURM plugin has been developed according to version 22.05.8 of SLURM, thus this specific version has been installed to manage the cluster. Version 0.2.0dev10 of the StreamFlow tool has been deployed. This version includes a specific plugin that enables the interaction with WARP through a polling mechanism as well as the modifications to add reservations when submitting workflows to the scheduler. This StreamFlow plugin also implements a basic exchange mechanism in order to synchronize workflows and steps' status with WARP.

5.1 Warming Up the Experimental Cluster

One critical aspect of the design and integration of resource allocation policies is to properly define a validation strategy. The natural choice for addressing this aspect should be applying the newly devised policy on a production machine, where real load conditions can happen. Getting access to production machines may be costly; nevertheless, cloud resources are a valid alternative. Given such an experimental virtual cluster, a mechanism to easily reproduce a real load becomes necessary; moreover, the mechanism should provide more control (in a statistical sense) over the generated load. To this end, in this work, we leveraged a dataset of 12,786 samples (*i.e.*, jobs with their features –, arrival times of submitted jobs (expressed in seconds), the wall-time duration (expressed in minutes and with a maximum allowed value of 60 minutes) and the number of required resources) gathered on a real supercomputer (actually a small subset of 16 nodes, each equipped with 36 cores, which fits more with the characteristics of our experimental virtual cluster), to train statistical models. Then, these models are used to generate a list of jobs to be submitted to the batch scheduler at different point in time (following the learned statistic). Specifically, we fit 3 independent probability distributions to emulate the job submissions distribution: *i*) *Exponential* distribution for interarrivals with rate $\lambda \approx 1.30\, e^{-3}$, on average a new HPC job is submitted every 12 minutes; *ii*) *Multinoulli* distribution for the number of nodes required by a job, with possible outcomes of 1,2,3 or 4 nodes; and *iii*) *Gaussian mixture* model for walltime durations, with 2 components to catch the bi-modality of the distribution, the corresponding means are 12 and 60 minutes. These three models are the core of the generator function that enables control over the jobs' distribution that is being created. The warming-up procedure then, uses these distributions to define the number of jobs (and their requirements) to be submitted to the compute nodes, thus reproducing a realistic initial workload.

6 AN EXAMPLE WITH A TURBINE DESIGN WORKFLOW

After setting up the simulation environment and finding a way to deploy a plausible load to the virtual cluster, we identified a workflow to test the WARP resource allocation logic. This workflow is a reduced and simplified version of a more complex one, which is used to support the aerodynamic design of aero-engine turbines. In detail, the workflow is made of two steps: LES step identifies a SLURM job that runs a simplified Large-Eddy Simulation (LES). LES, a mathematical model utilized in computational fluid dynamics to simulate turbulence phenomena, produces numerical data that serve as input to the High-Performance Data Analytics (HPDA) step. HPDA receives numerical data and analyses them. Thus, there is a direct dependence between the two steps of the workflow. Table 1 reports the resource requirements of the two steps.

Step ID	Nodes	Exec. time	Deployment
LES	2	00:15:00	SLURM
HPDA	5	00:30:00	SLURM

Table 1: Step requirements

When the user submits the workflow leveraging a classic WMS, this tool, following the step dependencies, tries to submit the LES step to the batch scheduler. If required resources are available, the corresponding job is executed. At the end of LES execution, the WMS proceeds with the submission of a new request to resources for the execution of the second step. Potentially, the second step must wait until enough nodes are available to trigger its execution. This can lead to a situation where single jobs of the workflow are executed in the minimum time but the total amount of time spent to execute the entire workflow is much greater than the sum of the two job execution times, due to the waits among the jobs. Thanks to the introduction of the WARP in the workflow execution flow, we can introduce more determinism to the scheduling of jobs, thus avoiding the waiting time among workflow steps. In this case, the user directly interfaces with the WARP through its CLI. Similarly to what happens with canonical WMS, the user submits the workflow providing the configuration files. WARP supports the input of workflows following the CWL format. Additionally, a configuration file related to the deployment platform and implementation of the workflow is needed, as CWL is implementation-agnostic. The benefits resulting from the introduction of the WARP into the workflow orchestration process emerge predominantly when the load on the cluster is particularly heavy, *i.e.*, when there are jobs in the running state and jobs in a pending state, waiting for node allocation. This is a representative situation of a production HPC cluster.

```
ReservationName=wp5user_250 StartTime=2024-02-23T10:57:45 EndTime=2024-02-23T11:11:45 Duration=00:14:00
   Nodes=compute[1-2] NodeCnt=2 CoreCnt=16 Features=(null) PartitionName=g100_usr_prod Flags=PURGE_COMP=00:00:01
   TRES=cpu=16
   Users=wp5user Groups=(null) Accounts=(null) Licenses=(null) State=INACTIVE BurstBuffer=(null) Watts=n/a
   MaxStartDelay=(null)

ReservationName=wp5user_251 StartTime=2024-02-23T11:12:45 EndTime=2024-02-23T11:26:45 Duration=00:14:00
   Nodes=compute[1-2,4,9-10] NodeCnt=5 CoreCnt=40 Features=(null) PartitionName=g100_usr_prod Flags=PURGE_COMP=00:00:01
   TRES=cpu=40
   Users=wp5user Groups=(null) Accounts=(null) Licenses=(null) State=INACTIVE BurstBuffer=(null) Watts=n/a
   MaxStartDelay=(null)
```

Figure 2: Reservations listed from scontrol

Table 2 shows the jobs in the running and pending state at T_0. Data are retrieved directly from the batch scheduler and jobs

named 'sleep_job' are simulated with the algorithm described in section 5.1. Reservations can also be listed running the SLURM command *scontrol show reservation*. The output is reported in Figure 2. Given the workflow structure, the reservations are allocated in sequence. The first reservation is assigned to the LES step. The second reservation starts at the end of the previous one and it is assigned to the HPDA step.

Job ID	Name	Status	Nodes	Node list
1600	sleep_job	R	2	[1-2]
1602	sleep_job	R	1	4
1605	sleep_job	R	2	[5-6]
1606	sleep_job	R	2	[7-8]
1607	sleep_job	R	1	3
1608	sleep_job	R	1	9

Table 2: Cluster status at T_0

Running nearly at the same moment two instances of the same workflow, one with WARP and the other one without WARP, we can find that LES_w, managed with the WARP, is assigned correctly to the reserved resources (Job ID=1611). On the other end, LES_s (LES job managed directly by the WMS, *i.e.*, StreamFlow) is executed immediately as there were enough free nodes in the cluster but the $HPDA_s$ job (Job ID=1613) is stuck in a pending state (see Table 3).

Job ID	Name	Status	Nodes	Node list
1600	sleep_job	R	2	[1-2]
1602	sleep_job	R	1	4
1605	sleep_job	R	2	[5-6]
1606	sleep_job	R	2	[7-8]
1607	sleep_job	R	1	3
1608	sleep_job	R	1	9
1611	sbatch	PD	2	(Reservation)
1613	sbatch	PD	5	(Resources)

Table 3: Cluster status at T_1

At T_1, LES_s job has been executed and does not appear in the list of running/pending jobs, while there is $HPDA_s$ that is waiting to be allocated. According to the internal allocation logic of the WARP, implemented in the optimizer, reservations are scheduled considering the expected load of the cluster. When the load is below 50% of the total number of nodes in the cluster, the reservations are scheduled. This parameter can be further adjusted. At T_2, according to the *start time* of the reservation, LES_w changed its status to *running*. After the execution of LES_w, $HPDA_w$ appears as a *pending* job, according to the *start time* of its reservation. At T_3, $HPDA_w$ starts simultaneously with the beginning of the second reservation (see Table 4), and few seconds later also $HPDA_s$ is executed. Due to the simplified version of *LES*, data processed are very limited and *HPDA* jobs last few seconds. Finally, Table 5 reports an overall comparison between the workflow managed directly by WMS and the one managed by the WARP.

As described in this section, the execution of the two workflows demonstrated the benefits of the WARP. Indeed, according to the results collected, the introduction of the advanced-reservation mechanism allows to reduce the waiting time among jobs of the same workflow, optimizing the user experience, and providing more determinism for the workflow execution. WARP optimized workflow has a makespan of 16 minutes with respect to the 46 minutes of the non-optimized one. Moreover, the proposed approach should facilitate the management of interactive jobs where present in complex workflows, as users know in advance the starting and ending time and date of the reservation related to that specific step/job. We believe that improvements to the user experience deriving from knowing in advance *start time* of various jobs of a workflow are the real added-value of the WARP. Initial results reported in this work will be extended with other experiments, including more complex workflows, which foresee several interdependent and parallel steps.

Job ID	Name	Status	time	Nodes	Node list
1617	sleep_job	PD	0:00	2	(Priority)
1618	sleep_job	PD	0:00	1	(Priority)
1619	sleep_job	PD	0:00	2	(Priority)
1621	sleep_job	PD	0:00	2	(Priority)
1622	sleep_job	PD	0:00	1	(Priority)
1623	sleep_job	PD	0:00	1	(Priority)
1607	sleep_job	R	54:25	1	3
1613	sbatch	R	0:14	5	[5-8,11]
1620	sbatch	R	0:29	5	[1-2,4,9-10]
1614	sleep_job	R	37:07	1	[12-13]

Table 4: Cluster status at T_3

Step	Submitted_at	Starte_at	Ended_at
LES_w	10:26:49	10:57:45	11:05:49
LES_s	10:27:11	10:27:11	10:33:00
$HPDA_w$	11:05:53	11:12:45	11:13:23
$HPDA_s$	10:33:01	11:12:53	11:13:37

Table 5: Comparison between the two workflows

7 CONCLUSIONS

The ever growing compute capabilities of modern HPC systems opened the doors for new, more complex workflows, which pair large-scale numerical simulations with computations belonging to other domains (*e.g.*, IA model training and inference, in-situ data visualization, data analytics, etc.). Optimal resource allocation becomes of paramount importance to keep the performance pace. In this work, we presented the WARP, a tool developed in the context of the ACROSS EuroHPC-JU funded project. WARP integrates with a workflow execution engine and a batch scheduler to make (optimal) resource allocation simple, by means of an on-demand reservation mechanism. Through a concrete example we demonstrated the effectiveness of the proposed tool, leaving the scalability study as a future research activity.

8 ACKNOWLEDGMENTS

This work is supported by the EuroHPC-02-2019 ACROSS project, grant agreement No 955648.

REFERENCES

[1] Enis Afgan, Dannon Baker, Marius Van den Beek, Daniel Blankenberg, Dave Bouvier, Martin Čech, John Chilton, Dave Clements, Nate Coraor, Carl Eberhard, et al. 2016. The Galaxy platform for accessible, reproducible and collaborative biomedical analyses: 2016 update. *Nucleic acids research* 44, W1 (2016), W3–W10.

[2] Dong H Ahn, Ned Bass, Albert Chu, Jim Garlick, Mark Grondona, Stephen Herbein, Helgi I Ingólfsson, Joseph Koning, Tapasya Patki, Thomas RW Scogland, et al. 2020. Flux: Overcoming scheduling challenges for exascale workflows. *Future Generation Computer Systems* 110 (2020), 202–213.

[3] Dong H Ahn, Jim Garlick, Mark Grondona, Don Lipari, Becky Springmeyer, and Martin Schulz. 2014. Flux: A next-generation resource management framework for large HPC centers. In *2014 43rd International Conference on Parallel Processing Workshops*. IEEE, 9–17.

[4] Malcolm Atkinson, Sandra Gesing, Johan Montagnat, and Ian Taylor. 2017. Scientific workflows: Past, present and future. , 216–227 pages.

[5] Sarah Cohen-Boulakia, Khalid Belhajjame, Olivier Collin, Jérôme Chopard, Christine Froidevaux, Alban Gaignard, Konrad Hinsen, Pierre Larmande, Yvan Le Bras, Frédéric Lemoine, et al. 2017. Scientific workflows for computational reproducibility in the life sciences: Status, challenges and opportunities. *Future Generation Computer Systems* 75 (2017), 284–298.

[6] Iacopo Colonnelli, Barbara Cantalupo, Ivan Merelli, and Marco Aldinucci. 2020. StreamFlow: cross-breeding cloud with HPC. *IEEE Transactions on Emerging Topics in Computing* 9, 4 (2020), 1723–1737.

[7] Rafael Ferreira Da Silva, Rosa Filgueira, Ilia Pietri, Ming Jiang, Rizos Sakellariou, and Ewa Deelman. 2017. A characterization of workflow management systems for extreme-scale applications. *Future Generation Computer Systems* 75 (2017), 228–238.

[8] Marco D'Amico and Julita Corbalan Gonzalez. 2021. Energy hardware and workload aware job scheduling towards interconnected HPC environments. *IEEE Transactions on Parallel and Distributed Systems* (2021).

[9] Ewa Deelman, Karan Vahi, Gideon Juve, Mats Rynge, Scott Callaghan, Philip J Maechling, Rajiv Mayani, Weiwei Chen, Rafael Ferreira Da Silva, Miron Livny, et al. 2015. Pegasus, a workflow management system for science automation. *Future Generation Computer Systems* 46 (2015), 17–35.

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

[11] Scott Haines. 2022. Workflow Orchestration with Apache Airflow. In *Modern Data Engineering with Apache Spark: A Hands-On Guide for Building Mission-Critical Streaming Applications*. Springer, 255–295.

[12] Hameed Hussain, Saif Ur Rehman Malik, Abdul Hameed, Samee Ullah Khan, Gage Bickler, Nasro Min-Allah, Muhammad Bilal Qureshi, Limin Zhang, Wang Yongji, Nasir Ghani, et al. 2013. A survey on resource allocation in high performance distributed computing systems. *Parallel Comput.* 39, 11 (2013), 709–736.

[13] Johannes Köster and Sven Rahmann. 2012. Snakemake—a scalable bioinformatics workflow engine. *Bioinformatics* 28, 19 (2012), 2520–2522.

[14] Steven H Langer, Brian Spears, J Luc Peterson, John E Field, Ryan Nora, and Scott Brandon. 2016. A HYDRA UQ workflow for NIF ignition experiments. In *2016 Second Workshop on In Situ Infrastructures for Enabling Extreme-Scale Analysis and Visualization (ISAV)*. IEEE, 1–6.

[15] Ji Liu, Esther Pacitti, Patrick Valduriez, and Marta Mattoso. 2015. A survey of data-intensive scientific workflow management. *Journal of Grid Computing* 13 (2015), 457–493.

[16] Fabrizio Marozzo, Francesc Lordan, Roger Rafanell, Daniele Lezzi, Domenico Talia, and Rosa M Badia. 2012. Enabling cloud interoperability with compss. In *Euro-Par 2012 Parallel Processing: 18th International Conference, Euro-Par 2012, Rhodes Island, Greece, August 27-31, 2012. Proceedings 18*. Springer, 16–27.

[17] Alessio Netti, Cristian Galleguillos, Zeynep Kiziltan, Alina Sîrbu, and Ozalp Babaoglu. 2018. Heterogeneity-aware resource allocation in HPC systems. In *High Performance Computing: 33rd International Conference, ISC High Performance 2018, Frankfurt, Germany, June 24-28, 2018, Proceedings 33*. Springer, 3–21.

[18] Tom Oinn, Mark Greenwood, Matthew Addis, M Nedim Alpdemir, Justin Ferris, Kevin Glover, Carole Goble, Antoon Goderis, Duncan Hull, Darren Marvin, et al. 2006. Taverna: lessons in creating a workflow environment for the life sciences. *Concurrency and computation: Practice and experience* 18, 10 (2006), 1067–1100.

[19] Miriam Peta. 2021. Reliability-aware resource management for HPC workload. (2021).

[20] JL Peterson. 2018. Machine learning aided discovery of a new nif design. *Lawrence Livermore National Laboratory* (2018).

[21] Michael A Salim, Thomas D Uram, J Taylor Childers, Prasanna Balaprakash, Venkatram Vishwanath, and Michael E Papka. 2019. Balsam: Automated scheduling and execution of dynamic, data-intensive hpc workflows. *arXiv preprint arXiv:1909.08704* (2019).

[22] Chiara Vercellino, Alberto Scionti, Giuseppe Varavallo, Paolo Viviani, Giacomo Vitali, and Olivier Terzo. 2023. A machine learning approach for an HPC use case: The jobs queuing time prediction. *Future Generation Computer Systems* 143 (2023), 215–230.

[23] Dali Wang, X Luo, Fengming Yuan, and Norbert Podhorszki. 2017. A data analysis framework for earth system simulation within an in-situ infrastructure. *Journal of Computer and Communications* 5, 14 (2017).