https://doi.org/10.5194/egusphere-2025-1104
Preprint. Discussion started: 20 May 2025

Evaluating the Impact of Task Aggregation in Workflows with Shared Resource Environments: use case for the MONARCH application

Manuel G. Marciani[1,2], Miguel Castrillo[1], Gladys Utrera[2,1], Mario C. Acosta[1], Bruno P. Kinoshita[1], and Francisco Doblas-Reyes[1,3]

[1]Barcelona Supercomputing Center (BSC), Plaça Eusebi Güell, 1-3, Barcelona,Spain
[2]Universitat Politècnica de Catalunya (UPC), Carrer Jordi Girona, 1-3, Barcelona, Spain
[3]Institució Catalana de Recerca i Estudis Avançats (ICREA), Barcelona, Spain

Correspondence: Manuel G. Marciani (manuel.gimenez@bsc.es) and Mario C. Acosta (mario.acosta@bsc.es)

Abstract. High Performance Computing (HPC) is commonly employed to run high-impact Earth System Model (ESM) simulations, such as those for climate change. However, running workflows of ESM simulations on cutting-edge platforms can take long due to the congestion of the system and the lack of coordination between current HPC schedulers and workflow manager systems (WfMS). The Earth Sciences community has estimated the time in queue to be between 10% to 20% of the runtime in climate prediction experiments, the most time-consuming exercise. To address this issue, the developers of Autosubmit, a WfMS tailored for climate and air quality sciences, have developed wrappers to join multiple subsequent workflow tasks into a single submission. However, although wrappers are widely used in production for community models such as EC-Earth3, MONARCH, and Destination Earth simulations, to our knowledge, the benefits and potential drawbacks have never been rigorously evaluated. In addition, with portability in mind, the developers proposed to wrap depending on the entitlement of the user to the machine. In the widely utilized Slurm scheduler, this factor is called fair share. The objective of this paper is to quantify the impact of wrapping on queue time and understand its relationship with the fair share and the job's CPU and runtime request. To do this, we used a Slurm simulator to reproduce the behavior of the scheduler and, to recreate a representative usage of an HPC platform, we generated synthetic static workloads from data of the LUMI supercomputer and a dynamic workload from a past flagship HPC platform. As an example, we introduced jobs modeled after the MONARCH air quality application in these workloads, which we tracked their queue time. We found that, by simply joining tasks, the total runtime of the simulation reduces up to 7%, and we have indications that this value is larger in reality. This saving translates to absolute terms in at least eight days less wasted in queue time for half of the simulations from the IS-ENES3 consortium of CMIP6 simulations. We also identified a high inverse correlation, of -0.87, between the queue time and the fair share factor.

1 Introduction

High-impact Earth System Model (ESM) simulations are normally executed in cutting-edge High Performance Computing (HPC) resources. In these environments, they typically entail several steps, including model compilation, data movement,

https://doi.org/10.5194/egusphere-2025-1104
Preprint. Discussion started: 20 May 2025

model execution, and post-processing. Furthermore, the execution of the model is often divided into multiple segments, that we refer as "chunks" of simulated time, which is done to comply with job submission constraints imposed by the system administrators. This results in workflows of up to thousands of individual tasks, as is the case with Destination Earth digital twin's (Hoffmann et al., 2023). To handle all of this complexity, ESM simulations are automatized with workflows.

Given the high cost associated of running such simulations, the Earth Sciences community has always been looking for ways of reducing the time-to-solution of their runs. The traditional front is to minimize the execution time of the most computationally intensive code within the model, such as the work of Irrmann et al. (2022).

However, with the recent findings of Acosta et al. (2024), there has been a growing awareness of considering the entire execution of the workflow, taking into account not only the runtime of the most demanding part of it, but also the time spent queuing for resources and post-processing, with possible failures. The authors found that the runs of the IS-ENES3 consortium of CMIP6 simulations, one of the most important worldwide exercises that contributes to the assessments and reports of the International Panel on Climate Change, had from 10% to 20% of the runtime of the simulation wasted in queue. And they conclude that this overhead is mainly from the workload of the machine, rather than the size of the simulation.

Modern HPC machines utilize a workload manager, or scheduler, to manage all of the jobs of the users. This software is responsible for allocating resources and queuing jobs when these are unavailable. The development of schedulers is an active area of research within the HPC scheduler community, focusing on optimizing job scheduling policies to minimize queue time, while keeping a high utilization of the machine (Brucker, 2007). Some examples of these policies are: first come, first served, least processing time, and priority-based. This last one is called *multifactor* in the ubiquitous Slurm workload manager (Jette and Wickberg, 2023).

Under the multifactor policy, jobs are ordered decreasingly by an integer called "priority", which is computed from the job request, time in queue, and the user's fair share. The purpose of this last factor, which is typically prioritized by system administrators, is to balance the responsiveness of the machine by favoring users who have not used their quota of the resources while penalizing those who overuse the machine. Besides the job's priority, Slurm also aims to maximize system utilization by implementing a technique called backfill (Srinivasan et al., 2002). This technique allows jobs with less priority to use free spots in the machine, as long as it does not interfere with the jobs ahead of them.

Observing the impact of fair share on the priority, the developers of Autosubmit (Manubens-Gil et al., 2016), a Workflow Manager System (WfMS) designed for climate and air-quality applications, have implemented a task aggregation technique called wrappers. The idea is that Autosubmit aggregates, or wraps, multiple subsequent and/or concurrent tasks into a single job submission to save the users from queuing multiple jobs under a low priority.

Task aggregation was also identified elsewhere from Earth Sciences, but with a different motivation. For example, in the paper by Abhinit et al. (2022), the authors claimed that "submitting these [tasks] alone may be an option (...) the site administrators may not allow" because of the "volume of tasks." They have also identified the same queue issues for long simulations, "queuing and startup overhead for a large number of small jobs could accumulate to prohibitively long total workflow execution time."

https://doi.org/10.5194/egusphere-2025-1104
Preprint. Discussion started: 20 May 2025

Their solution is the same as the one we evaluate in this work: "to submit a single pilot job to the scheduler that merely acquires the resources and launches an external workflow automation tool to scale-out within the single allocation."

Currently in our field, wrappers are used to reduce the queue time in the MONARCH (Klose et al., 2021) application and in the climate model EC-Earth3 (Döscher et al., 2022). They are also employed in the Destination Earth digital twin, to both reduce queue time and ensure compliance with platform's policy regarding maximum jobs queuing per user.

Although Autosubmit users have reported positive impact, there is a lack of understanding of the reasons and conditions under which aggregating reduces queue time. Therefore, in this work, we tested if wrapping tasks together reduces queue time and if the fair share is the most important factor in reducing queue time.

To test both theses, we measured queue time by simulating the HPC environment using a Slurm simulator (Ana Jokanovic, Marco D'Amico, and Julita Corbalan, 2018). We modeled a representative workload of an HPC platform and use the workflow of air-quality application. Then, we ran under the same conditions a simulation with wrappers and without.

From the results obtained, we found that combining tasks into one submission can reduce the queue time by an average of 7% of the total workflow runtime. However, this can be higher based on the machine's state. To put it into context of the CMIP6 runs of the IS-ENES3, this reduction would equate to more than eight days less of queue time for half of the runs. Also, we found a high inverse correlation, of -0.87, between the fair share factor and the time in queue.

Our study provides a quantification of the reduction of queue time achieved by aggregating tasks of air quality applications that use HPC resources on congested platforms. This approach, can also be applied on other simulation with temporal constraints that required a large amount of resources for a sustained time. Our results help to advance the understanding, from the user side, on how to optimize the submission in order to reduce the total queue time of their workflows.

This paper is organized as follows: the background section 2 explains the highlights of the scheduling policy. The methodology 3 provides a detailed explanation of the experimentation. This includes how we introduce the workflow onto the workload files, how we control the fair share, and the software we used and implemented to run the simulations. The results section 4 exposes the output of the experiments. This leads to the discussion 5 chapter, where we comment our findings and the relation between fair share and queue time. Finally, in the conclusions, we evaluate if our findings support our theses.

2 Background

In this section, we start with an overview on Slurm's scheduling in subsection 2.1 because of its wide adoption across HPC platforms. Then, we explain why and how wrappers are used in subsection 2.3.

2.1 Scheduling Algorithms of Slurm

When a job is submitted, Slurm will try to allocate resources to it immediately. If there are not any resources available, it will be put in a queue. This queue is sorted in descending order according to the *priority* integer, which is updated periodically.

Slurm's scheduling design consists of two coordinated algorithms that traverse the ordered list of queued jobs and attempt to allocate resources to them. The main algorithm tries to schedule as many jobs as possible from the queue. If it fails because

https://doi.org/10.5194/egusphere-2025-1104
Preprint. Discussion started: 20 May 2025

of lack of resources, it breaks the loop and sleeps. There are two options for the second algorithm: *builtin* and *backfill*. The first option works like the main algorithm. The second option is the backfill algorithm (Srinivasan et al., 2002). This algorithm is an optimization technique that allows jobs with lower priority to be scheduled before higher-priority jobs, as long as they do not interfere with the start time of the higher priority jobs. So, the smaller the job in terms of wallclock and/or CPUs requested, the more likely it is of being backfilled. So, without considering the effects of the backfill algorithm, jobs with higher priority will be scheduled earlier.

2.2 Fair Share Factor

The fair share factor is the quantification of a user's right to the machine, which is used to balance the responsiveness of the machine among all users according to their entitlement to it. Slurm uses this factor, along with time in queue, size, quality of service (QoS), and partition, to compute the priority under the *multifactor* policy, which is the default one (SchedMD, 2022). In Slurm, factors are floating point values between zero and one, which are then multiplied by the corresponding weights defined by the system administrators.

There are two ways to calculate the fair share factor: the *Classic* method and the *Fair Tree* method (SchedMD, 2019). We will focus on the Fair Tree method because it is the one used by default.

In the Fair Tree algorithm, users and accounts are associated to an account, which is typically created for each project. Both users and accounts are given an integer value called *RawShare*, usually related to their budget for the machine. Then, the usage of both user and account is tracked in the *RawUsage* integer. Succinctly, this algorithm does a depth-first traversal of the tree of users and accounts, ordering decreasingly the users or accounts of each account by the quotient of their *RawShare* and *RawUsage*. It then evaluates if it is an account or a user. If it is an account, it does a recursive call on it. If it is a user, it assigns the fair share and returns.

The fair share is assigned by taking into account the total number of users, N, and the position in which the user was evaluated, i. The computation is just $\frac{N-i+1}{N}$, which means that the lower bound, that is, the worst fair share possible, is $\frac{1}{N}$, when $i = N$.

This method implements shared accountability of the users for their entitlement to the machine. For example, if there are two sibling accounts, A and B, and A has a higher level fair share than B, then all users and accounts under A will have a higher fair share than B. So users are impacted not only by their own usage, but also the sum of their peers' usage.

2.3 Wrappers

In a shared HPC environment, queuing for resources is ever so frequent (Patel et al., 2020), and users have a limited impact on the priority of their jobs given the importance of fair share.

To reduce the time-to-solution of an Earth System Model (ESM) simulation's workflow, Autosubmit developers came up with a technique called task aggregation or wrapping. The idea is to improve execution throughput under a low fair share scenario by submitting fewer, larger jobs.

https://doi.org/10.5194/egusphere-2025-1104
Preprint. Discussion started: 20 May 2025

There are two basic categories of wrappers: vertical and horizontal. In the vertical wrappers, workflow tasks are executed sequentially. In the horizontal wrappers, tasks run in parallel. Both types submit tasks to the HPC platform in the same job, using the same allocation. There is also a combination of the two types: vertical-horizontal and horizontal-vertical. The vertical-horizontal is made up of multiple vertical wrappers running concurrently. Similarly, the horizontal-vertical is a single job made of multiple subsequent horizontal wrappers.

In this work we will focus on the vertical wrappers.

3 Methods

In this section, we explain the methodology employed in this study. First, in subsection 3.1, we provide a description of the HPC platforms we simulate and the hardware we used to run the simulations. Then we give an overview of the simulator in subsection 3.2. We cover the scheduling policy utilized in our experimentation in subsection 3.3. In subsection 3.4, we explain how we carried out simulations with synthetic static modeled after the (LUMI, 2024) supercomputer to address the modern job demands. In the subsection 3.5, we ran a dynamic simulation utilizing a real workload from a long decommissioned system to have a representative daily usage pattern. Finally, for modeling the workflow, we use the allocated CPUs and runtime of the job running the Nonhydrostatic Multiscale Model on the B-grid (NMMB) within the MONARCH application (Klose et al., 2021).

We perform a two-fold experimentation because we wanted to capture both a representative daily usage pattern (arrival times) and modern HPC job demands (requested CPUs, wallclock, user and group identification).

3.1 HPC Platforms and Execution Framework

We wanted to model a large, shared, and general purpose scientific machine. For that, we gathered data from the Large Unified Modern Infrastructure (LUMI) supercomputer (LUMI, 2024) operated by the EuroHPC and the Finnish Center for Science (CSC). This is a flagship, modern, highly utilized, and scientific platform, within the top five machines according to the TOP500 list (Strohmaier et al., 2023). Also, given the sheer quantity of HPC resources, it is the center pillar for many European projects, such as Destination Earth (Hoffmann et al., 2023). We used this data to build synthetic workloads because we are confident that the job geometry (allocated CPUs and runtime) is representative.

For the dynamic workloads, we utilized the workload from the decommissioned Curie machine, which was operated by the *Commissariat a l'Energie Atomic*, available in Feitelson and Tsafrir (2019) online repository.

Regarding the execution framework of this work, all simulations were carried out using our Laptop computer, with an Intel i5-1135G7 and 16GB of RAM.

3.2 Slurm Simulator

We employed the BSC Slurm simulator (Ana Jokanovic, Marco D'Amico, and Julita Corbalan, 2018) to recreate the scheduler behavior. This simulator is made of the same executables that make up Slurm, with only minimal changes to control the pace of the time and the input data.

https://doi.org/10.5194/egusphere-2025-1104
Preprint. Discussion started: 20 May 2025

Option	Value
default_queue_depth	10,000 $jobs$
defer	True
sched_interval	60 s
bf_interval	60 s
bf_max_time	30 s
bf_resolution	1,800 s
bf_window	10,080 min
bf_continue	True
PriorityMaxAge	10 $days$

Table 1. Main and backfill scheduler configuration (SchedMD, 2022). Configurations starting with *bf* apply to the backfill algorithm. These are used in MareNostrum 4 and we use them in all of our experiments.

We implemented a prologue to the original run script (G. Marciani, 2024b) to use Slurm's account manager tool, *sacctmgr*, in order to include users and accounts, which were determined from the input workload file and configured before running the simulation. All users and accounts were given the same entitlement to the machine, in other words, the same *RawShares* value.

Since the usage impacts the fair share value (SchedMD, 2019), it was important for us to ensure that all users have nil registered before the simulation. So we developed a Docker (Merkel, 2014) image (G. Marciani, 2024c) that ensures that the environment is clean, on top of being lightweight compared with using a virtual machine.

Finally, the simulator outputs a list of all the jobs that it processed, with the submit, start, end, and priority upon finish.

3.3 Slurm Configuration

In all of our experiments, we used MareNostrum 4's (BSC-CNS, 2023) scheduling configuration and priority weights, which is included in our repository (G. Marciani, 2024b). The scheduling configurations are specified in Table 1, where those configurations starting with *bf* apply to the backfill algorithm.

This policy uses the *multifactors* policy (SchedMD, 2023), where the fair share and age factor weights are $100,000$ and the size factor is $10,000$. The fair share factor is computed with the *Fair Tree* algorithm and with *backfill* as secondary scheduler. We only adapted the total number of physical cores available for each platform we model: 360,448 for LUMI and 93,312 for CEA-Curie.

We chose this system's policy because it is from a large and general purpose machine, besides it provided HPC resources to many groups across Spain and Europe, making it a highly demanded system.

https://doi.org/10.5194/egusphere-2025-1104
Preprint. Discussion started: 20 May 2025

3.4 Static Workloads

Static workloads are those where all jobs in the description are submitted at the same time. We made this decision to simplify the modeling, since we disregard the complex modeling of the arrival time (Cirne and Berman, 2000). We chose the number of jobs to be generated so that we distress the system but still within plausible bounds.

With this workload, we are unable to send workflow tasks with dependencies because of its single submission design. Instead, we are interested in understanding how the different factors from the scheduling play a role in the queue time. So we take a single workflow task, and we vary its request, that is, its allocated CPUs, runtime, and the user's fair share. Then we track its wait time in the queue.

In this section, we explain how we analyzed and fit distributions to data from the LUMI supercomputer (LUMI, 2024), in subsection 3.4.1, and we describe how we set up the experimentation in subsection 3.4.2.

3.4.1 Workload Generator

We generated workloads by fitting a log normal distribution to both the runtime and the allocated CPUs to the observed distribution using a script we developed G. Marciani (2024e). We achieved the following fitted values, in SciPy's nomenclature, for the allocated CPUs: $loc = -9.8930e-1$, $scale = 1.9930e+2$, and $s = 1.7195$; and, for the runtime: $loc = 9.5007e-1$, $scale = 1.6506e+2$, and $s = 2.6766$; with a sum squared error of, $1.5213e-07$ and $1.4533e-09$ respectively. In Figures 1 and 2, we have respectively the log allocated CPUs and log runtime cumulative distribution of the observed and the generated data, in orange dashed line and blue solid respectively.

Because the log normal distribution is unbounded, to generate the allocated CPUs we need to truncate it. Finally, for both the runtime and allocated CPUs the generated values were rounded to the nearest lower integer.

Once we generated a job's characteristics, that is its runtime and allocated CPUs, we assigned it to one of the hundred users uniformly at random. Similarly, we also assigned users to accounts. If a task was attributed to a user without an account, we assigned it to one out of a hundred uniformly at random. We generated ten workloads to account for the variability of the job's characteristics. And we drew 1,000 jobs for each workload because it provided a distressing yet still plausible scenario.

3.4.2 Experimental Design

We added a single workflow task to the ten generated synthetic static workloads in order to track its queue time. With the goal of measuring the impact of the different wrapper configurations, we made the geometry of this single task be a multiple of 1,800 seconds of runtime and 96 CPUs. We take this values from executions in MareNostrum 4 of the NMMB model of the MONARCH application (Klose et al., 2021).

In order to control the fair share, given that all the jobs of the workload are submitted at the same time, we preceded the simulation with a batch of "dummy" jobs so that all users have usage recorded. Otherwise, all users would have nil usage, therefore maximum fair share, and it would effectively remove the fair share from the scheduling. We set all the users "dummy"

https://doi.org/10.5194/egusphere-2025-1104
Preprint. Discussion started: 20 May 2025

Statistic	CPUs	Runtime (s)	$CPU \cdot s$
Mean	774.2	3,952.7	3.420701e+06
Std	4,493.4	19,617.9	1.392048e+08
Min	1	0	0
25%	48	23	3.533000e+03
50%	256	125	3.942400e+04
75%	1,024	902	2.150400e+05
Max	325,120	4,604,100	9.505604e+10

Table 2. Number of allocated CPUs, runtime, and core seconds statistics on the dataset captured in LUMI. 25% refers to the first quartile, 50%, the median, and 75%, the third quartile.

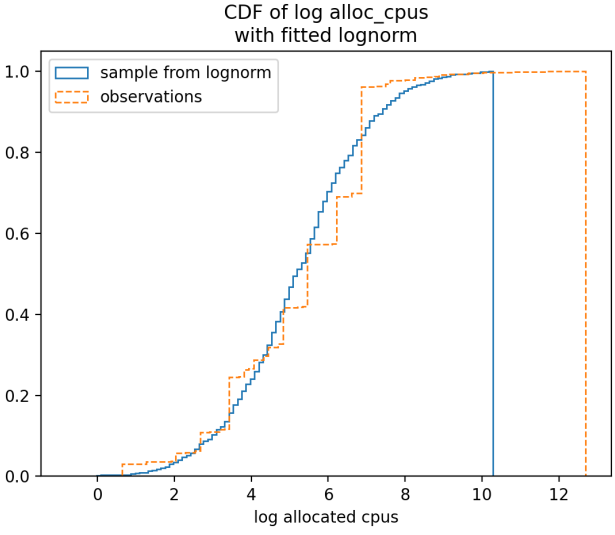

Figure 1. Cumulative distribution function of the log normal distribution for the allocated CPUs at LUMI. The orange dashed line is the log observed cumulative allocated CPUs and the blue solid line is a random sample generated with the parameters $loc = -9.8930e - 1$, $scale = 1.9930e + 2$, and $s = 1.7195$.

https://doi.org/10.5194/egusphere-2025-1104
Preprint. Discussion started: 20 May 2025

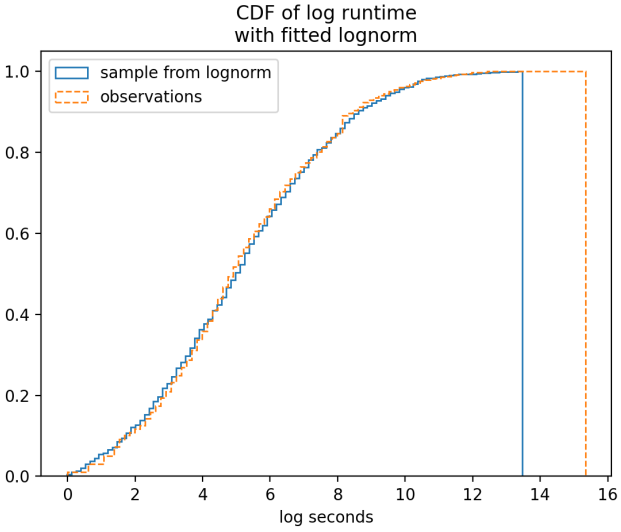

Figure 2. Cumulative distribution function of the log normal distribution for the runtime at LUMI. The orange dashed line is the observed cumulative log runtime of jobs, with runtime different than 0, and the blue solid line is the cumulative distribution of a random sample generated with the parameters $loc = 9.5007e-1$, $scale = 1.6506e+2$, and $s = 2.6766$.

submission runtime to be proportional to the synthetically generated usage, except for the user employed with launching the workflow, that we chose its usage to control its fair share.

We tested every combination of 1,800, 3,600, 7,200, and 12,600 seconds of runtime and 96, 192, 384, 672, and 1,152 CPUs. For the fair share of the user launching the workflow 2.2, we tested 0.1, 0.2, 0.25, 0.3, 0.5, 0.7, 0.75, 0.8, and 0.9.

3.5 Dynamic Workload

Opposed to the static workloads, jobs are submitted at different times in dynamic workloads. This subsection explains how we set up the experiment for them. In section 3.5.1 we cover the choice of workload, and in section 3.5.2 we cover how we set up the experiment.

3.5.1 Workload Choice

We used the CEA-Curie clean version 2 from Feitelson and Tsafrir (2019) repository, which only considers those jobs submitted after a major upgrade in the machine and removes those which were certainly badly logged, i.e., reported running for far too long. This dataset is one of the largest publicly available workloads for a machine following the criteria we want.

We explored the workload in search of periods of congestion, which accumulate normally on Thursdays and Fridays on this machine and last a few days. We selected one week of the trace, between 9/6/2012 and 15/6/12. The resulting workload file is found at the code snippet alongside with the script to add the workflow (G. Marciani, 2024a).

https://doi.org/10.5194/egusphere-2025-1104
Preprint. Discussion started: 20 May 2025

Label	Submission Instant
A.1	14/6/2012 at 10
A.2	14/6/2012 at 15
A.3	14/6/2012 at 20
B.1	15/6/2012 at 10
B.2	15/6/2012 at 15
B.3	15/6/2012 at 20

Table 3. Submission instants of the workflow and their corresponding label.

3.5.2 Experimental Design

To test if wrappers improve the time-to-response, we considered a seven chunk split simulation of the MONARCH application, with the tasks requesting 96 CPUs and taking 1,800 seconds to execute, typical values of its execution in MareNostrum 4.

The simulator does not have support for dynamic submission times, i.e., only submit a constrained job when its dependency has finished, as it would be the case in real life with Autosubmit. Therefore, we had to specify the submission time on the workload file, prior to knowing when the tasks will be scheduled, and consequently finish.

Therefore, to minimize the waiting time of a constrained task from adding too much queue time, increasing its age factor, we defined its submission time as the instant its predecessor would have finished if it did not have any waiting time. That is, the second task is submitted 1,800 seconds after the first one; the third, 3,600 seconds after the first one; and so on and so forth.

In order to control the fair share of the user executing the workflow, we measured the usage up until the submission instant of all users by executing the workload with no workflow added to it. With this utilization, and taking into consideration that all accounts have the same entitlement to the resources, the fair share will be roughly one minus the percentile of the distribution of usage of the accounts. For instance, to have a fair share of 0.2 we need to match the utilization of the 80th percentile of the usage of the accounts.

We tested multiple fair share values: the worst, in this case is 0.01 – and the reason is explained in subsection 2.2 –, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, and the best, which is 1.0.

Then we considered the unwrapped case, in which all the tasks are launched individually, and the wrapped case, where we created a single job adding the runtime of all of them. In the case, that means a job of length 12,600 seconds and 96 CPUs. We simulated both unwrapped and wrapped cases, at six different submission times: at 10, 15 and 20 of both Thursday 14th and Friday 15th.

In Table 3, we label these submission instants and plot them in Figure 3 as black vertical dashed lines along with the evolution of in-queue and in-use resources in orange dashed and in blue solid lines, respectively, from a simulation of the original trace with no workflow added to it. The dashed blue line is the total number of CPUs of the CEA-Curie machine. We observe two clear orange dashed peaks indicating the daily usage cycle and the blue solid line with a clear maximum, which is the total available CPUs.

https://doi.org/10.5194/egusphere-2025-1104
Preprint. Discussion started: 20 May 2025

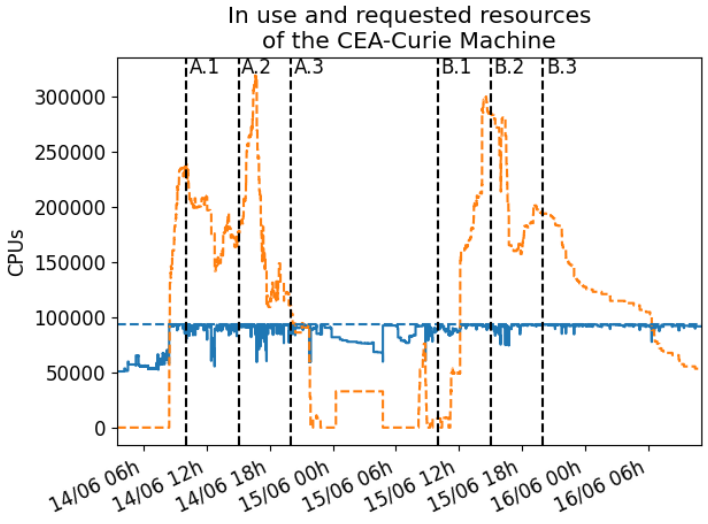

Figure 3. Simulated usage, in blue solid, and in queue resources, in orange dashed lines, from the untouched CEA-Curie workload for the week from 9/6/2012 to 15/6/12. The vertical black dashed line indicates the instant of submissions with labels following Table 3 and the blue dashed horizontal line is the total number of CPUs of the machine.

4 Results

In this section, we analyze the results of the runs of both types of experiments: the static workloads in subsection 4.1 and the dynamic workload in subsection 4.2.

Full results tables were omitted due to space constraints, but they are available in the Zenodo platform G. Marciani (2024f).

4.1 Static Workloads

We compute the correlation of the runtime, allocated CPUs, and fair share with the average, maximum, and minimum of both the queue time and the priority upon finishing the job, respectively T_q and P, in Table 4. We observe that the fair share is the dominant factor on the waiting time, with a clear inverse correlation of -0.87 between the average wait time and fair share. Moreover, we observe the independence of the allocated CPUs and runtime with respect to the queue time.

Additionally, in Figure 4 we plot the average wait time across all ten experiments for every job configuration in function of the fair share factor, where we see an exponential reduction in queue time as we increase the fair share, but we observe that its impact is smaller for the lower fair share values, visible from the spread of the each of the lines representing a job configuration. After all job configuration's merge into one. We fit an exponential model (G. Marciani, 2024h), where a is the exponent constant and b is the multiplicative one.

As for the priority upon finishing, we observe in Table 4 how the fair share is almost completely positively (0.99) correlated with the average, maximum, and minimum priority. The rest of the factors, allocated CPUs and runtime, show no correlation.

https://doi.org/10.5194/egusphere-2025-1104
Preprint. Discussion started: 20 May 2025

	Runtime	CPUs	Fair share
$max\ T_q$	0.0229	-0.0161	-0.8379
$min\ T_q$	0	0	-0.8572
$\overline{T_q}$	0.0048	0.0007	-0.8720
$max\ P$	0.0018	0.00623	0.9911
$min\ P$	-0.0002	0.00026	0.9929
$\overline{P}$	-0.0010	0.00128	0.9952

Table 4. Correlation table of the job and fair share with queue time and priority. T_q is the time in queue and P is the priority upon finishing of the job. An overline indicates the average across the ten experiments and max and min are the maximum and minimum observed across the 10 experiments of the particular quantity.

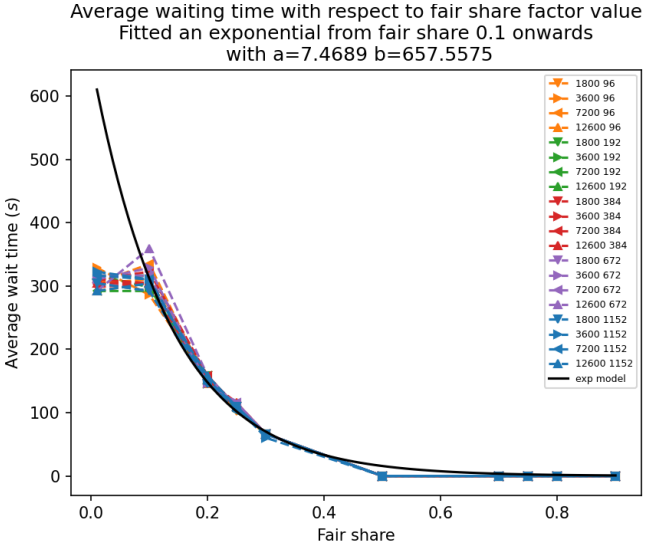

Figure 4. Average of the queue time across all experiments with respect to the fair share value. Each color represents a different runtime from 1,800, 3,600, 7,200, and 12,600 seconds, and each line style represents a different allocated CPUs configuration from 96, 192, 384, 672, and 1,152.

Analogous to the wait time, we plot in Figure 5 the average across all ten experiments of the priority in function of the fair share, for each and every combination of allocated CPUs and runtime. There, we see how the priority grows linearly with respect to the fair share, with the largest deviations in the low fair share spectrum.

https://doi.org/10.5194/egusphere-2025-1104
Preprint. Discussion started: 20 May 2025

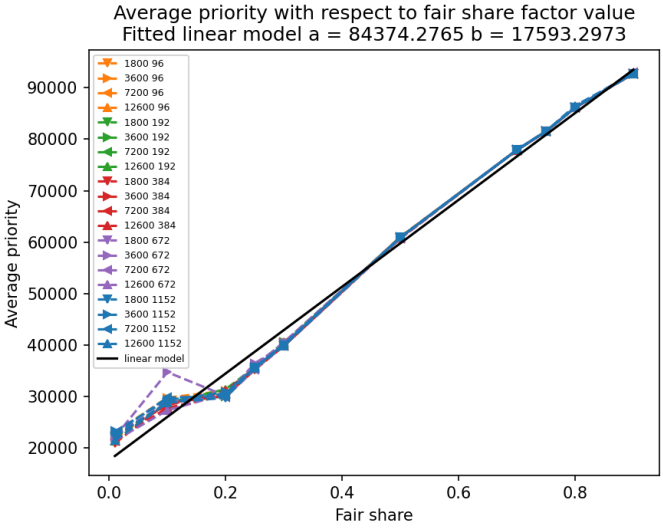

Figure 5. Average of the priority time across all experiments with respect to the fair share value. Each color represents a different runtime from 1,800, 3,600, 7,200, and 12,600 seconds, and each line style represents a different allocated CPUs configuration from 96, 192, 384, 672, and 1,152.

4.2 Dynamic Workload

We gathered all six instants and made a box plot of the difference in time-to-response, i.e., the difference between submission time and end of the workflow, between the unwrapped and the wrapped executions. Thus, a positive value indicates that the wrapped workflow tracked less queue time than the unwrapped counterpart.

In Figure 6, we see that the difference is positive on average, indicated by the green triangle. Also, the median, indicated by the orange line, is also always positive. However, we observe fliers where the unwrapped outperformed the wrapper. This is the case for the worst fair share, 0.4, and 0.6.

5 Discussion

We achieved reduction on queue time across all fair shares values, in the dynamic results, as seen in Figure 6. We highlight the maximum reduction that we observed, which was a 7% decrease in queue time relative to the total workflow runtime. This supports the thesis that the reduction is caused by avoiding multiple submissions under a low fair share. Therefore, we recommend to aggregate tasks under this low fair share setting, of less than 0.4.

Moreover, this 7% figure could be even greater if we take into account the workload of the machine we used, the CEA-Curie, had only two weekly days of congestion. A current flagship system is normally always congested, as seen, for example, in the dashboard Riken-CCS (2025) of the Japanese flagship machine, Fugaku.

https://doi.org/10.5194/egusphere-2025-1104
Preprint. Discussion started: 20 May 2025

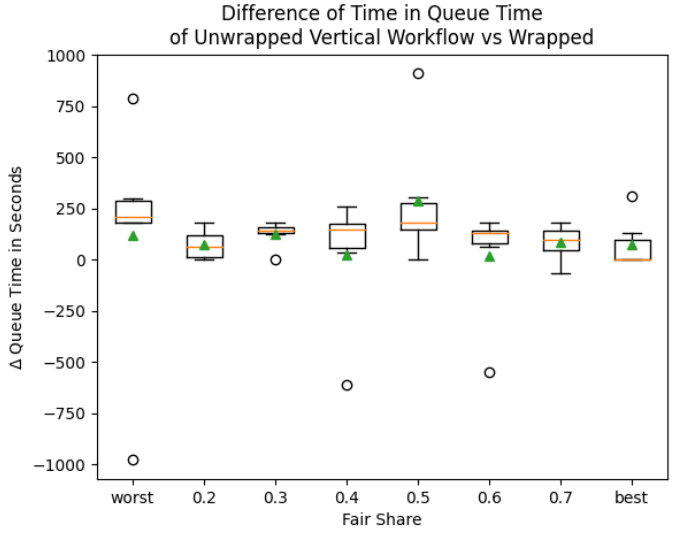

Figure 6. Box plot of the difference of the unwrapped queue time with respect to the wrapped. Green triangle indicates mean, orange horizontal line the median. Circles indicate fliers, which are instants of submissions that surpass 1.5 times the interquartile difference.

In the case of the negative outliers, that is, the instances where the unwrapped workflow stood less in queue than the wrapped, these are due to the backfill algorithm. As we increase the length of the job, it is less likely that the schedule would have free spots to execute, so the job has to wait to be scheduled by its priority.

As for the static results, where we captured current job requests modeled after the LUMI supercomputer, these give us an idea of what are the meaningful factors that immediately impact after submission under a distressed environment. What we see is a total correlation of the queue time with respect to the fair share, and, even stronger, an exponential relationship. This means that with a fair share of just 0.5, it is enough to have more priority than all the other jobs in the queue, and therefore to be scheduled immediately.

This exponential relation comes from the way we set the fair share to all the synthetically generated users. Since their prior usage was set to be proportional to the generated workload, their fair share followed a log-normal distribution.

Regarding job requests in static executions, we observe in Table 4 that both runtime and allocated CPUs are independent of queue time. This is explained because these simulations are short compared with the dynamic ones, not allowing for the age factor to build up, and neither the backfill algorithm to be noticeable.

Finally, we see the expected linear relationship of the priority with the fair share in Figure 5. This comes from the weighted sum that Slurm uses to compute the priority. We also have the slope of the line we fit, 84,374, almost matching the weight of the fair share taken from MareNostrum 4, of 100,000.

As is the case for the queue time, the job request is not correlated with the priority. This is due to the aforementioned shortness of the simulations and the relatively small application request with the large system. If we take the maximum any of our tracked jobs was in queue, 372 seconds, this adds just 61 to the priority from the age factor. And if we do the same for our

https://doi.org/10.5194/egusphere-2025-1104
Preprint. Discussion started: 20 May 2025

largest request in CPUs, 1152, it adds just 32 to the priority due to the size factor. Compared with a fair share of just 0.01 that adds 1000 to the priority, both the age and the size factors are marginal.

6 Conclusions

The time in queue of the simulations is a growing issue for those users utilizing highly congested machines. And it is even worse for those executing simulations with vertical workflows, where workflow tasks have to wait until their dependency is met. This paper analyzes for the first time, to the best of our knowledge, a simple but powerful solution to mitigate this issue, which is to aggregate tasks into a single submission to be sent to the HPC platform.

We measured the impact of aggregating tasks by running two simulations under the same conditions with a Slurm simulator, one wrapping and another with all the tasks independently submitted. We used the MONARCH air quality application as a reference for the application. To have both modern job requests and realistic behavior on the usage of the machines, we performed two experiment types: one with synthetic static workloads modeled after a current flagship system, where all the jobs are submitted at the same time, and another with a real dynamic workload, as recorded in a production machine.

We tied this impact with a key scheduling factor employed by system administrators of Slurm: the fair share. This factor quantifies the responsiveness of the machine to the user, and is shared with the group.

Our findings support the thesis that aggregating tasks reduces queue time. We observed a maximum difference in queue time between unwrapped and wrapped of 7% of the total runtime of the workflow. If we put this saving in terms of the 10% to 20% queue overhead reported by Acosta et al. (2024), that would be a reduction of the queue time of 35% to 70% percent. This result supports our thesis that with a low fair share, we submit fewer jobs, and hence fewer tasks of the workflow get constrained due to priority.

With the static experiments, we see how a fair share factor of just 0.5 is enough for the job to be executed immediately. This is not as clear in the dynamic results, but we observe that with a higher fair share, the impact of using wrappers decreases.

Finally, this paper addresses the issue of queue time, which is particularly acute for the climate community, but it may be shared with other communities running their applications on congested HPC machines. We expose the important parts of the inner workings of the scheduling algorithm of the most popular workload manager, Slurm, and relate them to the request in terms of CPUs and wallclock of the workflow. This work is of interest to all those who see the need to optimize their time-to-solution by providing recommendations on how to submit their tasks and the rationale as to why it is beneficial.

Code and data availability. The Slurm simulator code is available in https://earth.bsc.es/gitlab/mgimenez/ces_slurm_simulator under GPL2 license. The exact version of the simulator used to produce the results in this paper is archived on Zenodo (G. Marciani, 2024b).

The Docker image to run the Slurm simulator is available in https://earth.bsc.es/gitlab/mgimenez/docker-ubuntu-ces-slurm-sim under GPLv3 license. The exact version used in this paper is archived on Zenodo (G. Marciani, 2024c).

https://doi.org/10.5194/egusphere-2025-1104
Preprint. Discussion started: 20 May 2025

Analysis scripts for the workload and results are available, respectively, in https://earth.bsc.es/gitlab/mgimenez/scripts-hairball/-/snippets/128 and https://earth.bsc.es/gitlab/mgimenez/scripts-hairball/-/snippets/131 under GPLv2. The exact version used in this paper is available on Zenodo, respectively, G. Marciani (2024e) and (G. Marciani, 2024h).

Script to include workflow to Curie dynamic trace is available in https://earth.bsc.es/gitlab/mgimenez/scripts-hairball/-/snippets/125. The exact version used in this paper is available on Zenodo under GPLv2 (G. Marciani, 2024a).

The results of the simulations are made available in the Zenodo platform G. Marciani (2024f).

The original data and the input workload files, for both static and dynamic simulation, are made available on the Zenodo platform, respectively, G. Marciani (2024g) and G. Marciani (2024d).

Due to its sensitivity, data gathered from the Lumi supercomputer is not publicly available.

Author contributions. MGM has developed the methodology, the Docker image of the BSC Slurm simulator, the Python library for managing the Standard Workload Manager, ran the simulations, performed the analysis, and wrote the paper. MC has conceptualized the work, supervised the work, and revised the manuscript. GU and MCA have supervised the work and reviewed the manuscript. BPK he has reviewed the manuscript. FDR has provided funds for the development of the work.

Competing interests. The authors declare that they have no conflict of interest.

Acknowledgements. This work received support from grant CEX2021-001148-S-20-5 funded by MICIU/AEI/10.13039/501100011033 and by FSE+. Mario Acosta is supported from the Spanish National Research Council through OEMES (PID2020-116324RA-I00).

The authors declare they used DeepL in order to improve readability of the manuscript in all sections. After using this tool, the authors reviewed and edited the content as needed and take full responsibility for the content of the published article.

https://doi.org/10.5194/egusphere-2025-1104
Preprint. Discussion started: 20 May 2025

References

Abhinit, I., Adams, E. K., Alam, K., Chase, B., Deelman, E., Gorenstein, L., Hudson, S., Islam, T., Larson, J., Lentner, G., et al.: Novel proposals for FAIR, automated, recommendable, and robust workflows, in: 2022 IEEE/ACM Workshop on Workflows in Support of Large-Scale Science (WORKS), pp. 84–92, IEEE, 2022.

Acosta, M. C., Palomas, S., Paronuzzi Ticco, S. V., Utrera, G., Biercamp, J., Bretonniere, P.-A., Budich, R., Castrillo, M., Caubel, A., Doblas-Reyes, F., Epicoco, I., Fladrich, U., Joussaume, S., Kumar Gupta, A., Lawrence, B., Le Sager, P., Lister, G., Moine, M.-P., Rioual, J.-C., Valcke, S., Zadeh, N., and Balaji, V.: The computational and energy cost of simulation and storage for climate science: lessons from CMIP6, Geoscientific Model Development, 17, 3081–3098, https://doi.org/10.5194/gmd-17-3081-2024, 2024.

Ana Jokanovic, Marco D'Amico, and Julita Corbalan: Evaluating SLURM Simulator with Real-Machine SLURM and Vice Versa, Performance Modeling, Benchmarking and Simulation of High Performance Computer Systems (PMBS18) At: ACM/IEEE Supercomputing 2018, 2018.

Brucker, P.: Scheduling Algorithms, Springer, Berlin, Germany, 5 edn., 2007.

BSC-CNS: Blog post, https://www.bsc.es/marenostrum/marenostrum, 2023.

Cirne, W. and Berman, F.: Adaptive selection of partition size for supercomputer requests, in: Job Scheduling Strategies for Parallel Processing: IPDPS 2000 Workshop, JSSPP 2000 Cancun, Mexico, May 1, 2000 Proceedings 6, pp. 187–207, Springer, 2000.

Döscher, R., Acosta, M., Alessandri, A., Anthoni, P., Arsouze, T., Bergman, T., Bernardello, R., Boussetta, S., Caron, L. P., Carver, G., Castrillo, M., Catalano, F., Cvijanovic, I., Davini, P., Dekker, E., Doblas-Reyes, F. J., Docquier, D., Echevarria, P., Fladrich, U., Fuentes-Franco, R., Gröger, M., Hardenberg, J. V., Hieronymus, J., Karami, M. P., Keskinen, J. P., Koenigk, T., Makkonen, R., Massonnet, F., Ménégoz, M., Miller, P. A., Moreno-Chamarro, E., Nieradzik, L., Van Noije, T., Nolan, P., O'donnell, D., Ollinaho, P., Van Den Oord, G., Ortega, P., Prims, O. T., Ramos, A., Reerink, T., Rousset, C., Ruprich-Robert, Y., Le Sager, P., Schmith, T., Schrödner, R., Serva, F., Sicardi, V., Sloth Madsen, M., Smith, B., Tian, T., Tourigny, E., Uotila, P., Vancoppenolle, M., Wang, S., Wårlind, D., Willén, U., Wyser, K., Yang, S., Yepes-Arbós, X., and Zhang, Q.: The EC-Earth3 Earth system model for the Coupled Model Intercomparison Project 6, Geoscientific Model Development, 15, 2973–3020, https://doi.org/10.5194/gmd-15-2973-2022, 2022.

Feitelson, D. G. and Tsafrir, D.: Logs of real parallel workloads from production systems, https://www.cs.huji.ac.il/labs/parallel/workload/logs.html, 2019.

G. Marciani, M.: Scripts and Files to Add Workflow to Curie, Zenodo, https://doi.org/10.5281/zenodo.12801281, [software], 2024a.

G. Marciani, M.: BSC Computational Earth Sciences Slurm Simulator Tools, Zenodo, https://doi.org/10.5281/zenodo.12800999, [software], 2024b.

G. Marciani, M.: Docker Image of the Computational Earth Sciences Slurm Simulator, Zenodo, https://doi.org/10.5281/zenodo.12801138, [software], 2024c.

G. Marciani, M.: Wrapper Impact Workloads and BSC Slurm Simulator Output of Dynamic Traces from CEA Curie, Zenodo, https://doi.org/10.5281/zenodo.10623439, [dataset], 2024d.

G. Marciani, M.: Lumi Workload Analysis Script, Zenodo, https://doi.org/10.5281/zenodo.12801326, [software], 2024e.

G. Marciani, M.: Full Results from Simulations for Static and Dynamic Workloads Using BSC Slurm Simulator, Zenodo, https://doi.org/10.5281/zenodo.10818813, [dataset], 2024f.

G. Marciani, M.: Wrapper Impact Workloads and BSC Slurm Simulator Output of Static Traces based on Data from LUMI Supercomputer, Zenodo, https://doi.org/10.5281/zenodo.10624403, [dataset], 2024g.

https://doi.org/10.5194/egusphere-2025-1104
Preprint. Discussion started: 20 May 2025

G. Marciani, M.: Static Workload Results Analysis Scripts, Zenodo, https://doi.org/10.5281/zenodo.12801377, [software], 2024h.

Hoffmann, J., Bauer, P., Sandu, I., Wedi, N., Geenen, T., and Thiemert, D.: Destination Earth – A digital twin in support of climate services, Climate Services, 30, 100 394, https://doi.org/10.1016/j.cliser.2023.100394, 2023.

Irrmann, G., Masson, S., Maisonnave, E., Guibert, D., and Raffin, E.: Improving ocean modeling software NEMO 4.0 benchmarking and communication efficiency, Geoscientific Model Development, 15, 1567–1582, https://doi.org/10.5194/gmd-15-1567-2022, 2022.

Jette, M. A. and Wickberg, T.: Architecture of the Slurm Workload Manager, in: Job Scheduling Strategies for Parallel Processing. JSSPP 2023. Lecture Notes in Computer Science, vol 14283, edited by Klusáček Dalibor, Corbalán, J., and Rodrigo Gonzalo P, pp. 3–23, Springer Nature Switzerland, Cham, https://doi.org/10.1007/978-3-031-43943-8, 2023.

Klose, M., Jorba, O., Gonçalves Ageitos, M., Escribano, J., Dawson, M. L., Obiso, V., Di Tomaso, E., Basart, S., Montané Pinto, G., Macchia, F., et al.: Mineral dust cycle in the Multiscale Online Nonhydrostatic AtmospheRe CHemistry model (MONARCH) version 2.0, Geoscientific Model Development, 14, 6403–6444, 2021.

LUMI: Blog post, https://lumi-supercomputer.eu/, 2024.

Manubens-Gil, D., Vegas-Regidor, J., Prodhomme, C., Mula-Valls, O., and Doblas-Reyes, F. J.: Seamless management of ensemble climate prediction experiments on HPC platforms, in: 2016 International Conference on High Performance Computing and Simulation (HPCS), pp. 895–900, https://doi.org/10.1109/HPCSim.2016.7568429, 2016.

Merkel, D.: Docker: lightweight linux containers for consistent development and deployment, Linux journal, 2014, 2, 2014.

Patel, T., Liu, Z., Kettimuthu, R., Rich, P., Allcock, W., and Tiwari, D.: Job characteristics on large-scale systems: long-term analysis, quantification, and implications, in: SC20: International conference for high performance computing, networking, storage and analysis, pp. 1–17, IEEE, 2020.

Riken-CCS: Blog post, https://status.fugaku.r-ccs.riken.jp/, 2025.

SchedMD: Fair Tree Fairshare Algorithm, Blog post, https://slurm.schedmd.com/fair_tree.html, 2019.

SchedMD: Scheduling Configuration Guide, https://slurm.schedmd.com/sched_config.html, 2022.

SchedMD: Multifactor Priority Plugin, Blog post, https://slurm.schedmd.com/priority_multifactor.html, 2023.

Srinivasan, S., Kettimuthu, R., Subramani, V., and Sadayappan, P.: Selective reservation strategies for backfill job scheduling, in: Job Scheduling Strategies for Parallel Processing: 8th International Workshop, JSSPP 2002 Edinburgh, Scotland, UK, July 24, 2002 Revised Papers 8, pp. 55–71, Springer, 2002.

Strohmaier, E., Dongarra, J., Simon, H., and Meuer, M.: TOP500. The List., https://top500.org/, 2023.