H2020-JTI-EuroHPC-2019-1

REGALE: An open architecture to equip next generation HPC applications with exascale capabilities

Grant Agreement Number: 956560

D1.4

REGALE Evaluation

Final

Version: 1.0

Author(s): Georgios Goumas (ICCS), Nikos Triantafyllis (ICCS), Efstratios Karapanagiotis (ICCS), Eishi Arima (TUM), Andrea Bartolini (UNIBO), Mohsen Seyedkazemi Ardebili (UNIBO), Varvara Asouti (NTUA), Ioannis Kalogeris (NTUA), Ioannis Ledakis (UBI), Nikos Papageorgopoulos (UBI), Antonis Koukourikos (SCIO), Bruno Raffin (UGA), Pierre-Francois Detot (UGA), Olivier Richard (UGA), Alejandro Ribes (EDF), Mathieu Stoffel (ATOS), Federico Tesser (CINECA), Daniele Cesarini (CINECA), Evgenia Kontoleontos (Andritz), Rahul Banerjee (TWT), Philippe Mauri (TWT), Martin Obstbaum (TWT), Lluis Alonso (BSC), Julita Corbalan (BSC), Mattia Paladino (E4), Yiannis Georgiou (RYAX)

Date: 01.05.2024

Project and Deliverable Information Sheet

REGALE Project	**Project Ref. №:** 956560	
	Project Title: REGALE	
	Project Web Site: https://regale-project.eu	
	Deliverable ID: D1.4	
	Deliverable Nature: Report	
	Dissemination Level: PU *	**Contractual Date of Delivery**: 31/3/2024
		Actual Date of Delivery: 1/5/2024
	EC Project Officer: Matteo Mascagni	

* - The dissemination levels are indicated as follows: PU = Public, fully open, e.g. web; CO = Confidential, restricted under conditions set out in Model Grant Agreement; CI = Classified, information as referred to in Commission Decision 2001/844/EC.

Document Control Sheet

Document	**Title:** REGALE Requirements, Initial Architecture, and Evaluation Plan	
	ID: D1.4	
	Version: 1.0	**Status:** Final
	Available at: https://regale-project.eu	
	Software Tool: Google Docs	
	File(s): REGALE_D1.4_Evaluation_1.0.pdf	
Authorship	**Written by:**	Georgios Goumas (ICCS), Nikos Triantafyllis (ICCS), Efstratios Karapanagiotis (ICCS), Eishi Arima (TUM), Andrea Bartolini (UNIBO), Mohsen Seyedkazemi Ardebili (UNIBO), Varvara Asouti (NTUA), Ioannis Kalogeris (NTUA), Ioannis Ledakis (UBI), Nikos Papageorgopoulos (UBI), Antonis Koukourikos (SCIO), Bruno Raffin (UGA), Pierre-Francois Detot (UGA), Olivier Richard (UGA), Alejandro Ribes (EDF), Mathieu Stoffel (ATOS), Federico Tesser (CINECA), Daniele Cesarini (CINECA), Evgenia Kontoleontos (Andritz), Rahul Banerjee (TWT), Philippe Mauri (TWT), Martin Obstbaum (TWT), Lluis Alonso (BSC), Julita Corbalan (BSC), Mattia Paladino (E4), Yiannis Georgiou (RYAX)
	Contributors:	As above
	Reviewed by:	Andrea Bartolini (UNIBO), Bruno Raffin (UGA), Josef Weidendorfer (BADW-LRZ)
	Approved by:	Georgios Goumas (ICCS)

Document Status Sheet

Version	Date	Status	Comments
0.1	9/1/2024	Draft	Table of contents
0.2	2/2/2024	Draft	Initial text inserted
0.3	10/3/2024	Draft	Experimental setup described
0.4	10/4/2024	Draft	Initial evaluation data inserted
0.5	17/4/2024	Draft	Final evaluation data
0.6	26/4/2024	Draft	Ready for review
0.7	1/5/2024	Draft	Review comments integrated
1.0	1/5/2024	Final	Deliverable ready

Document Keywords

Keywords:	REGALE, HPC, Exascale, Software Architecture, Evaluation, Power Stack, Workflow Engines

Copyright notice:

© 2024 REGALE Consortium Partners. All rights reserved. This document is a project document of the REGALE project. All contents are reserved by default and may not be disclosed to third parties without the written consent of the REGALE partners, except as mandated by the European Commission contract 956560 for reviewing and dissemination purposes.

All trademarks and other rights on third party products mentioned in this document are acknowledged as owned by the respective holders.

Executive Summary

This deliverable provides the evaluation results across all the activities of the REGALE project, i.e. a) the REGALE prototypes (Integration Scenarios) and the REGALE library, b) the REGALE pilots and c) the activities related to the study and incorporation of sophistication in a state-of-the art supercomputing software stack. The evaluation process presented here is largely targeted to assess the status of the Strategic Objectives set by the project. To the Consortium's view, these objectives have been met to a large extent as validated by the experimental data provided in this document.

Table of Contents

1. Introduction

Throughout its implementation, REGALE dealt with two significant and inter-related challenges for the effective utilization of large-scale supercomputing facilities, i.e. a) the efficient use *system resources* translated to system throughput, application performance and energy efficiency, and b) the efficient use of *human resources* translated to the easy, automated and flexible use of supercomputing services.

To this end the project set a number of ambitious goals (i.e. Strategic Objectives, see Section 2) aligned with the aforementioned general vision. In this deliverable we present evaluation results from several activities within the project that aim to ultimately validate whether the objectives set by the project have been met. Since REGALE involves both quantitative and qualitative objectives, this deliverable focuses on providing results for the quantitative ones and provides a discussion (Section 8) on the qualitative ones.

The results presented are split in three categories aligned with the general activities and work packages of the project. Section 3 details the project objectives and provides an overview on their status at the completion of the project. In Section 4 and 5 we present results from the REGALE prototypes and the REGALE library that mainly target energy efficiency. In Section 6 we present the results from the integration of the REGALE pilots with the relevant workflow engines that target application performance and automation in the allocation of system resources. In Section 7 we present results from the activities of REGALE that targeted the injection of sophistication in modern HPC software stacks. Section 8 discusses briefly the qualitative objectives of REGALE and Section 9 concludes the deliverable.

2. Project Strategic Objectives

REGALE Strategic Objectives: REGALE envisions to meet the Strategic Objectives (SO) presented below.

Strategic Objective 1 (SO1): Effective utilization of resources. This strategic objective considers the large amount of resources available in exascale class machines and the resource footprints of both traditional and emerging applications. The improvement in resource utilization will indicatively translate to a combination of:

- **SO1.1: Improved application performance.** Better allocation of resources that considers the exact application footprint, data requirements, and control and data flows, will drastically improve performance for critical applications. This is especially the case for the next generation, workflow-based applications where one of the major problems is the highly suboptimal use of resources, leading to reduced performance, inability to scale, misuse of resources and consequent over-charging of end users.
- **SO1.2: Increased system throughput.** By taking global and elaborate decisions considering the entire mix of workloads to be executed in the supercomputer, we aim to significantly raise the system throughput, servicing more applications per day and ultimately increasing user satisfaction and system impact.
- **SO1.3: Minimized performance degradation under power constraints.** Power capping is a common mechanism to align supercomputer consumption with the power availability and charges of the supplier. In REGALE we will replace the current brute-force, performance-oblivious strategies by a set of sophisticated policies for dynamic adaptation to power envelopes without compromising application performance and system throughput.
- **SO1.4: Decreased energy to solution.** REGALE supports the operation of a supercomputer with energy consumption as a first class citizen. In this case we incorporate mechanisms and policies to minimize energy to solution if this is promoted by the operation policy.

Strategic Objective 2 (SO2): Broad applicability. This strategic objective has guided our architecture design and prototyping towards maximizing openness, platform independence, scalability, modularity, extensibility and simplicity, allowing for its implementation with various software modules, on any supercomputing platform, for the realization of SO1. In particular, this will be achieved through compatibility to relevant specifications and standards.

To assess if this SO is met, we validate the existence of the following key features:

- **Scalability:** The REGALE system should be able to operate in exascale setups and beyond.
- **Platform independence:** The REGALE system should be able to operate across all major architectures of large supercomputing facilities and be free of any vendor lock-in.
- **Extensibility:** The REGALE system should be extensible to any new feature or component that aligns to its open architecture.

Strategic Objective 3 (SO3): Easy and flexible use of supercomputing services. Widening the use of advanced computational and data facilities beyond the highly skilled traditional HPC users requires significant efforts on the side of the centers. In REGALE we release the developers and users of complex applications that originate from new industrial use cases from the extremely cumbersome task to finetune the execution of their application

on an exascale system. Moreover, we equip them with an easy-to-use set of tools to facilitate the development and deployment of their applications to exascale systems.

We validate this SO through the existence of the following key features:

- **Automatic allocation of resources:** Users of complex applications should not bother with the way their application is distributed on an exascale system.
- **Programmability:** Application developers should find the REGALE architecture and system easily accessible to develop and deploy their code(s).
- **Flexibility:** Applications should be able to execute under lightweight virtualization within the REGALE- enabled system.

3. REGALE Strategic Objectives and KPIs

The evaluation for REGALE is aligned to the Strategic Objectives of the project as described in the DoA and summarized in Table 3.1 together with the relevant KPIs and targets. The objectives of the project are both quantitative and qualitative. SO1 together with its sub-objectives SO1.1-SO1.4 target specific quantitative metrics relevant to resource utilization that influence performance, energy efficiency and combined metrics. SO2 is mostly relevant to the qualitative characteristics (functional requirements) of the REGALE library and SO3 is relevant to widening the use of supercomputing resources to more complex, next generation applications (REGALE pilots).

Table 3.1 also provides an overview of the status of each one of these objectives, together with a reference to the section of this deliverable where more information is provided. As mentioned, the current deliverable focuses on the quantitative objectives of REGALE. For the qualitative ones, the reader is directed to Section 8 for an overview and to other deliverables where more detailed information is provided.

Overall, based on the outcome of a rather extensive evaluation campaign, we may note that the Strategic Objectives of the project have been met.

Table 3.1: Summary of REGALE Strategic Objectives, relevant KPIs and status based on the evaluation results

Strategic Objective		Baseline	KPI	Target	Status
SO1	SO1.1: Improved application performance.	Execution in State -of-the-Art HPC systems	Quantitative: FLOPS or time to solution	20%	Met with the node and GPU co-scheduling schemes (Sections 7.1-7.2)
		Pilots before REGALE		2x	Met by pilots (Section 6.1 - 6.5)
	SO1.2: Increased system throughput.	State-of-the-Art HPC systems	Quantitative: jobs / hour	30%	Met with the node and GPU co-scheduling schemes (Sections 7.1-7.2)
	SO1.3: Minimization of performance degradation under the operation with power constraints.	State-of-the-Art HPC systems	Quantitative: Decrease throughput penalty	50% less penalty	Met with Tag-based Application-Aware Power Capping on a specific case - further experimentation planned (Section 4.1) Met with application-aware power capping with scheduler support (Section 4.3)
	SO1.4: Decreased energy to solution.	State-of-the-Art HPC systems	Quantitative: energy reduction	10%	Met with application-aware energy optimization (Section 4.2) Met with application-aware power capping with scheduler support (Section 4.3)
SO2	Scalability	n/a	Quantitative	exascale	Initial evidence for the REGALE library with preliminary experiments (Section 5) Initial evidence for energy optimization with simulation results (Section 4.3) Further experimentation is needed
	Platform independence	n/a	Qualitative	n/a	Met by the design of the REGALE library (Section 8 and Deliverable D3.3)
	Extensibility	n/a	Qualitative	n/a	Met by the design of the REGALE library (Section 8 and Deliverable D3.3)
SO3	Automatic allocation of resources	Traditional HPC application development / deployment	Qualitative	n/a	Met by: The integration of the pilots with the workflow engines (Deliverable D4.3) The integration of the machine learning models in production systems (Section 7.6)
	Programmability	Traditional HPC application development	Qualitative	n/a	Met by the integration of the pilots with the workflow engines (Deliverable D4.3)
	Flexibility	Traditional HPC application deployment	Qualitative	n/a	Met by the integration of the pilots 3 and 4 with the RYAX workflow engine (Deliverable D4.3)

4. Results of REGALE prototypes

The current state-of-practice in supercomputing operation includes systems that a) are completely energy/power-unaware and perform brute-force power capping[1] when needed, b) have included some form of energy awareness by monitoring and reporting energy consumption (again with brute-force power-capping) and c) systems that include some primitive energy/power-awareness and enforce some static energy policies. To the best of our knowledge supercomputing centers do not perform operational sophisticated energy management or power capping.

The prototypes of REGALE also called Integration Scenarios (see D3.3) primarily target SO1.3, and secondarily SO1.2 and SO1.4. They rely on an enhanced architecture that is able to monitor, decide and act appropriately in order to apply sophisticated power capping and energy control. The ultimate goal is to deliver the benefits of power capping (reduced energy and thermal effects, compliance to the current state of the power grid) while minimizing its potentially severe effects on application performance and system throughput. In the next paragraphs we provide evaluation results of the relevant Integration Scenarios.

REGALE value proposition: REGALE designs and implements a prototype system that, with the proper collaboration of various modules (monitors, node managers, job managers, RJMS), is able to apply sophisticated power capping and reduce its penalty in system throughput by 50%. Below, we provide results from three REGALE prototypes that work in this direction, i.e., a) Tag-based Application-Aware Power Capping, b) Application-aware energy optimization under a system power cap and c) Application-aware power capping with scheduler support.

4.1 Tag-based Application-Aware Power Capping

The rationale underlying the evaluation strategy for Tag-based Application-Aware Power Capping (TAAPC) is to assess how it performs when compared to the standard Fair-Sharing Power Capping (FSPC) policy, described in the below subsection. This evaluation is performed for different power budgets to enforce on the managed partition of compute nodes during the executions of several scheduled sets of jobs representative of typical supercomputing workloads.

Key Performance Indicator

Given the above, short summary of the evaluation strategy for TAAPC, it clearly appears that it strongly relates to the Strategic Objective SO1.3, which is representative of the fact that the main goal of TAAPC is to minimize the performance degradation induced by operating with power constraints.

Consequently, one Key Performance Indicator (KPI) seems perfectly suited to the quantitative evaluation of TAAPC, namely the job throughput (jobs/s). For a given scheduling of HPC jobs executed on a given partition of compute nodes under a given power budget, the associated objective is hence that the performance degradation (i.e. the decrease of the job throughput) induced by TAAPC is lesser than the one induced by FSPC. On the one hand, the baseline job throughput against which those performance degradations are evaluated is

[1] By brute-force power capping we refer to the process where the entire power-cap set for the data center is applied evenly to the server nodes disregarding the nature of the jobs running on the system.

associated with the execution of the same scheduling on the same partition of nodes, but this time without any power constraint enforced. On the other hand, FSPC is a power-capping strategy according to which all the nodes of the managed partition are power-capped to the same ratio of their nominal consumption. For instance, let's imagine a partition of 10 nodes. 5 of them are of model A, which draws 450W, and the other 5 nodes are of model B, which draws 300W. If the power envelope to enforce for the partition is 3375W (instead of the 3750W nominal power consumption), then FSPC consists in enforcing power caps on the 10 nodes to reduce their relative power consumptions by 10% (405W for a node of model A, and 270W for a node of model B).

Finally, numerically, the objective set for TAAPC at the start of the project was to induce 15% lower performance degradations than FSPC.

Experimental platform

The experimental platform used to evaluate TAAPC is an Atos on-premise supercomputing partition consisting of 12 BullSequana X440-A5 compute nodes (details below) with an Infiniband HDR100 interconnection network. The partition is managed by Slurm (version 23.11), and both Bull Energy Optimizer (BEO) and its module implementing TAAPC are installed on the management nodes. BEO features 1Hz out-of-band energy monitoring of the compute nodes, and has the capability to enforce power caps on the latter, also in an out-of-band fashion.

Details of the BullSequana X440-A5 compute blades:

- 2 sockets with AMD Epyc 7282 (2 x 16 cores) @ 2.80 GHz
- 128 GB DDR4 (8 x 16 GB) @ 3200 MHz
- Direct Liquid Cooling (DLC)

Evaluation protocol

Before describing the evaluation protocol, we define the notion of a "schedule of HPC jobs". In the following, it represents a fixed set of HPC jobs (i.e. of HPC applications being executed on sets of compute nodes), submitted following fixed and reproducible sequence and timeline. In other words, defining a "schedule of HPC jobs" is tantamount to defining an ordered sequence of job submissions. As a result, it makes it possible to evaluate the impact of a power management strategy on the job throughput associated with a given supercomputing workload.

And that is precisely the goal of this experimental protocol, in which several schedulings of HPC jobs are to be executed on the partition of compute nodes described in the previous section, with three different configurations regarding power management, for a range of power budgets:

1. Without enforcing any power caps;
2. With power caps enforced according to the FSPC policy;
3. With power caps enforced according to the TAAPC mechanism.

Here, the considered range of power budgets is defined with respect to the nominal power envelope required to operate all the compute nodes of the partition at their Thermal Design Power (TDP), called "aggregated TDP" (AggTDP) in the remaining of the document: from 100% to 80%, included, with a 5% step.

Regarding the considered schedules of HPC jobs, the below Table 4.1 presents synthetic information about them. Notably, it identifies each schedule by a codename, specifies the repartition of the executed applications between the three applicative tags, and adds additional information about the schedule when necessary. Below the table, a short rationale is given for every schedule. The fifth column contains the number of jobs which are executed right away at the start of the schedule and the number of jobs pending in the queue between parentheses.

As a reminder, the notion of “tag” is associated with the executed application, to qualify its dominating behavior. The first implementation of TAAPC defines three tags: (1) COMPUTE to identify compute-bound applications, (2) MEMORY to identify memory-bound applications, and (3) MIXED to identify applications which are not clearly dominated by one of the previous two behaviors. Hereinbelow, the pool of HPC applications used for the experiments described in this document consists of:

- COMPUTE: HPL and NPB.EP;
- MEMORY: HPCG, NEMO, and NPB.MG;
- MIXED: GROMACS, NPB.LU and NPB.FT.

Note that the workloads for those applications were designed/adjusted to induce an execution time without any power constraints of 10 to 15 minutes.

Table 4.1: Synthetic information about the schedules of HPC jobs involved in the evaluation of TAAPC.

Codename	COMPUTE	MIXED	MEMORY	# jobs (+ in queue)	Additional information
OCJUC	100%	0%	0%	1 (+0)	4 active nodes 8 idle nodes
HMHC	50%	0%	50%	4 (+0)	No waiting queue.
VNNQ	100%	0%	0%	7 (+0)	4 jobs on 1 node, 2 jobs on 2 nodes, and 1 job on 4 nodes
VNSQ	100%	0%	0%	7 (+3)	Same as VNNQ, plus 2 jobs on 1 node and 1 job on 2 nodes in the queue
VNBQ	100%	0%	0%	7 (+2)	Same as VNNQ, plus 1 job on 2 nodes and 1 job on 4 nodes in the queue
RWSCH	20%	60%	20%	7 (+8)	Alike production environment - Mixed workload + jobs in the queue

Rationale for OCJUC: The schedule consists of 1 COMPUTE job executed on 4 nodes while the 8 others are idle. On the one hand, for TAAPC, the expected behavior is that the idle nodes will be capped to their Minimal Operating Point (MOP), since they do not need any power to perform computation. As a result, it will free some power budget for the active

nodes executing the job, which will hence suffer only minimal performance degradations. On the other hand, the FSPC strategy will induce equally constraining power caps on all the nodes, whether active or idle. Consequently, the performance of the COMPUTE application executed by the four active nodes should be significantly degraded.

Rationale for HMHC: The schedule consists of 4 jobs, half of them tagged COMPUTE, and the other half tagged MEMORY. No waiting job in the queue, so, as soon as a job terminates, the nodes which were allocated to them are idle. The expected behavior for HMHC is to observe: (1) initially, a shift of the power budgets from MEMORY jobs to COMPUTE jobs, so that the latter could run with less performance degradations and complete sooner when compared to FSPC, (2) the power budgets being shifted back to the MEMORY jobs until their completions, (3) globally, the 4 jobs should complete sooner when compared with FSPC, and half of the partition should also be available sooner for subsequent jobs.

Rationale for VNNQ, VNSQ, and VNBQ: As described in Deliverable D2.3 “Final integration of sophisticated policies in the REGALE prototype”, the TAAPC mechanism enforces power capping rules within the tag-class of jobs starting with the jobs to which fewer nodes are allocated. This trio of evaluation scenarios aims at assessing the soundness of this design choice by using schedules consisting only of COMPUTE jobs with various numbers of allocated nodes and different types of queues (respectively, no jobs in queue, jobs with small numbers of nodes allocated in queue, jobs with large numbers of nodes in queue). The expected behavior is that:

- On average, jobs in the queue will have access to resources sooner with TAAPC when compared to FSPC since large jobs will have more power budget to complete sooner, hence freeing enough nodes to accommodate both large and small pending jobs;
- The increase of TtS induced by performance disparities under power capping (see D2.3 for more details) shall be lower for TAAPC when compared to FSPC ;
- As a result, at the global scale, the schedules shall complete sooner when the power budget is enforced by TAAPC when compared to FSPC.

Rationale for RWSCH: The schedule associated with this evaluation scenario aims to mimic real-world conditions on a production supercomputer, with mainly MIXED jobs, a varied number of nodes allocated to jobs, and a populated waiting queue of jobs. Thus, this evaluation scenario can be considered a functional and performance validation test for the TAAPC mechanism compared to the FSPC strategy.

Finally, we mention that each “experimental point” (i.e. the execution of a schedule for a configuration of the power management and a given power budget) was replicated seven times and that the data presented in the next section are the averages of those seven repetitions.

Limitations

The protocol defined in this document for the evaluation of TAAPC was only partially implemented due to several delaying events (e.g. faulty compute blades regarding power capping). However, the whole evaluation protocol is included in the roadmap of the PowerEfficiency team of Eviden. That is why TAAPC is still being worked on, and the protocol is to be fully carried out. The obtained results will be disseminated.

Below we provide a summary of the limitations of effectively implemented evaluation protocol to this date, when compared to the one described in the previous subsections:

- Only one power budget was evaluated, namely 85% of the AggTDP of the managed partition of nodes;
- Out of the set of evaluation scenarios presented previously, only OCJUC was implemented. Its results are discussed in the next subsection;
- Out of the corpus of applications to be used in the evaluation protocol, only HPL (CPU), was used for the limited implementation which results are presented below.

Results and discussions

OCJUC: As Table 4.2 shows, when it comes to enforcing an 85% AggTDP power budget for the OCJUC evaluation scenario, the TAAPC mechanism outperforms the FSPC strategy. As explained in the associated rationale, on the one hand by shifting the power budget from the idle nodes to the active nodes, TAAPC allows the active nodes to draw their TDPs. Consequently, the only performance degradations induced by TAAPC on the execution of the COMPUTE job are due to the time spans with boost frequencies enforced being limited. On the other hand, by capping the power consumption of the active nodes at 85% of their TDPs, FSPC induces roughly 13% of performance degradation. It means that TAAPC induces 94% less performance degradation than FSPC for the OCJUC scenario. This result satisfies the 15% KPI in this scenario associated with the evaluation protocol. To conclude, it is worth mentioning that this evaluation scenario is a special case representing an under-used supercomputer, which may happen in real-world production systems (e.g. at night, during holidays, …). And TAAPC is able, by design, to leverage such a situation when FSPC cannot.

Table 4.2: Mean durations (out of 7 executions) of the schedule associated with the evaluation scenario OCJUC. “Baseline” corresponds to executions without any power capping enforced. “TAAPC - φ%” corresponds to executions with a power budget of φ% of the AggTDP of the managed partition enforced by the TAAPC mechanism. Same meaning for “FSPC - φ%”, except that the power budget is enforced by the FSPC strategy.

Configuration	Duration (in seconds)	Relative increase of the TtS i.r.t “Baseline”
Baseline	785	0%
TAAPC - 85%	791	+0.765%
FSPC - 85%	886	+12.9%

4.2 Application-aware energy optimization under a system power cap

In this REGALE prototype, the Job Manager and the Node Manager optimize the power management state of the computational resources, selecting the right combination in correlation to the needs of the monitored applications. In this specific case, we decided to optimize the power consumption modifying the p-states during the run of the underlying application. We used as a benchmark the NAS IS, class E, run on the E4 subsystem composed of 4 nodes Ice Lake Intel(R) Xeon(R) Gold 6330 CPU @ 2.00GHz.

Experimental setup

The results obtained both with COUNTDOWN and EAR, each in respect to the monitoring system represented by EXAMON, were already presented in the deliverable D3.2

(paragraphs “COUNTDOWN - EXAMON” and “EAR - EXAMON”). Here we focus and report the results obtained from the specific integration between COUNTDOWN and EAR (described in the deliverable D3.2, paragraph “COUNTDOWN - EAR”). We note that, in this prototype, the *Monitor* provides feedback both to the system administrator and to the users on job efficiency and on metrics, through ad-hoc dashboards (see deliverable D3.2).

All the policies of EAR have been tested (*monitoring*, *min_energy*, *min_time*), with different starting p-states depending on the logic behind their internals. Moreover, two modalities of COUNTDOWN have also been applied: *monitoring* and *on*.

Under this last one, COUNTDOWN will apply the maximum frequency decided each time by EAR exiting the MPI events which triggered its threshold for the frequency modification, while the minimum frequency applied at the entrance of the MPI events mentioned, remained unchanged (the one stored in the file *cpuinfo_min_freq*).

Results

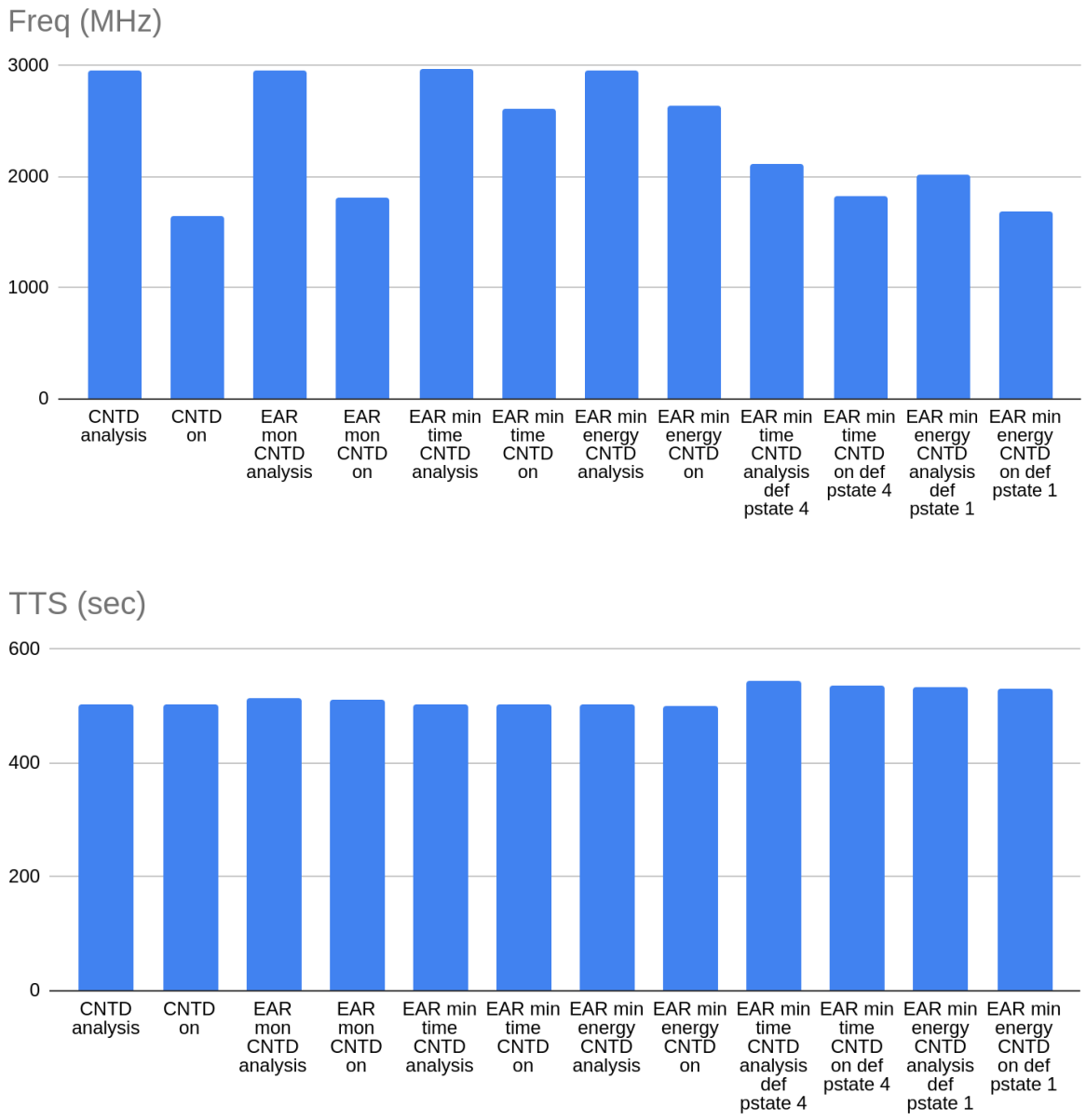
Table 4.3: KPIs obtained from the applications of the different policies both for EAR and COUNTDOWN.

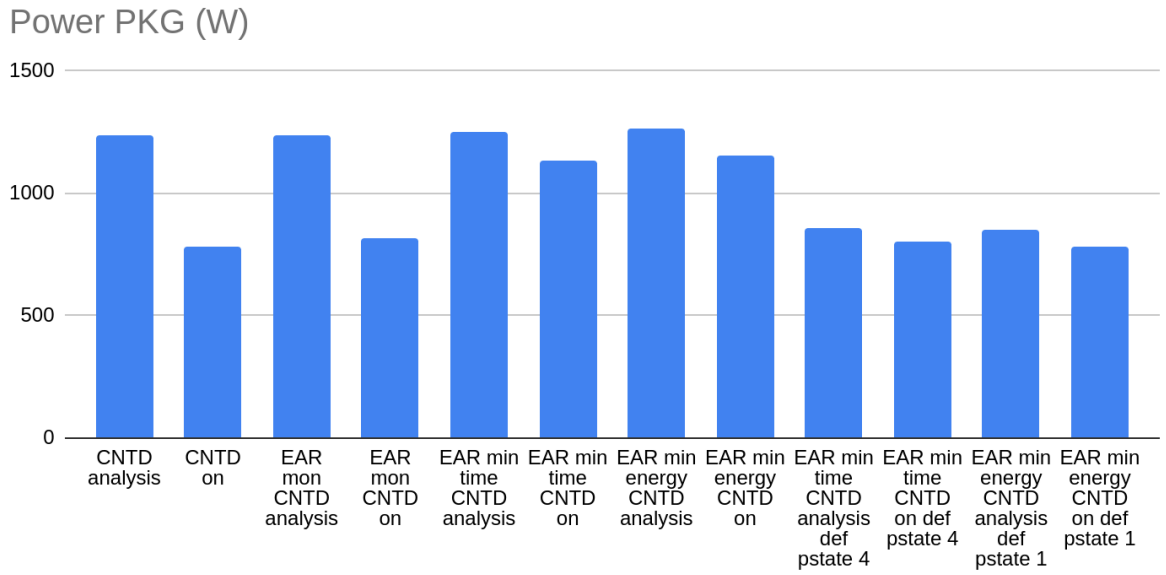
	AVG Freq	Energy PKG	Power PKG	TTS	Reduction in Energy	Reduction in Power	Increase in TTS
CNTD analysis	2963	622609	1237.22	503.234	-	-%	-%
CNTD on	1642	393735	781.05	504.107	36.76%	36.87%	0.17%
EAR mon CNTD analysis	2960	636774	1239.89	513.572	-2.28%	-0.22%	2.05%
EAR mon CNTD on	1810	418224	817.15	511.811	32.83%	33.95%	1.70%
EAR min time CNTD analysis	2970	627791	1247.97	503.049	-0.83%	-0.87%	-0.04%
EAR min time CNTD on	2608	570664	1133.53	503.438	8.34%	8.38%	0.04%
EAR min energy CNTD analysis	2963	635501	1265.81	502.051	-2.07%	-2.31%	-0.24%
EAR min time CNTD analysis def pstate 4	2116	465748	856.32	543.895	25.19%	30.79%	8.08%
EAR min time CNTD on def pstate 4	1822	431059	802.5	537.145	30.77%	35.14%	6.74%
EAR min energy CNTD analysis def pstate 1	2019	454263	852.06	533.136	27.04%	31.13%	5.94%
EAR min energy CNTD on def pstate 1	1693	413841	781.35	529.646	33.53%	36.85%	5.25%

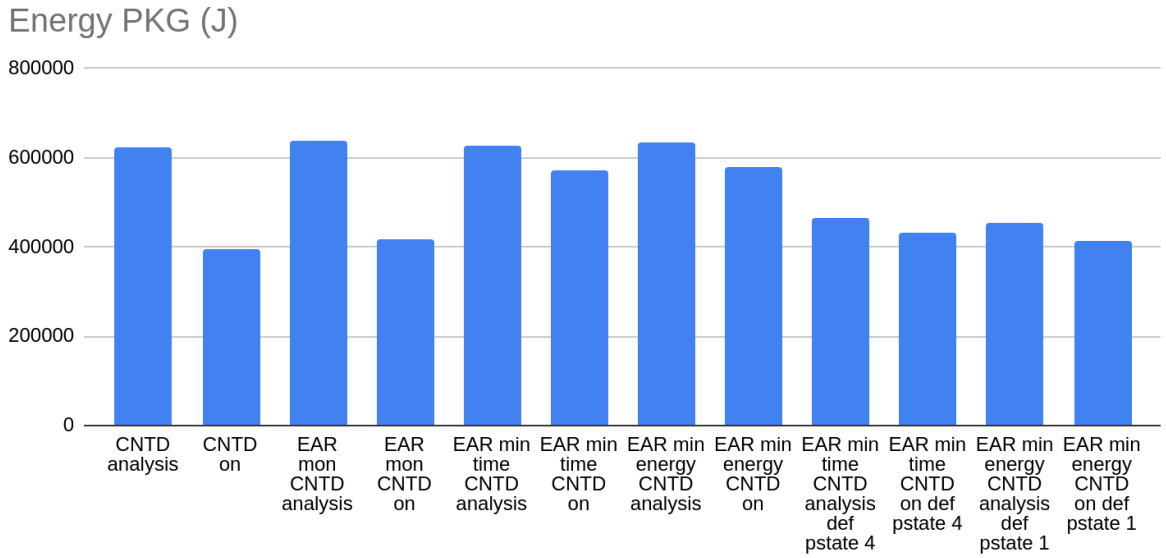
In Table 4.3 we report the results obtained for the different policies and modalities used in combination; We also present cases in which the starting p-states of EAR were modified enabling the turbo mode (the ones without the definition “def pstate 1” or “def pstate 4”, which actually represent the default behavior of the policies of EAR, regarding the maximum pstate available). In addition, in the table we report the KPIs used for the evaluation of this scenario: T.T.S. (Time To Solution) and Energy and Power consumed for reaching the final solution, with the corresponding increases or reductions, reported in percentage regarding the comparison case “CNTD analysis”. Labels used in the table: 1) CNTD analysis =

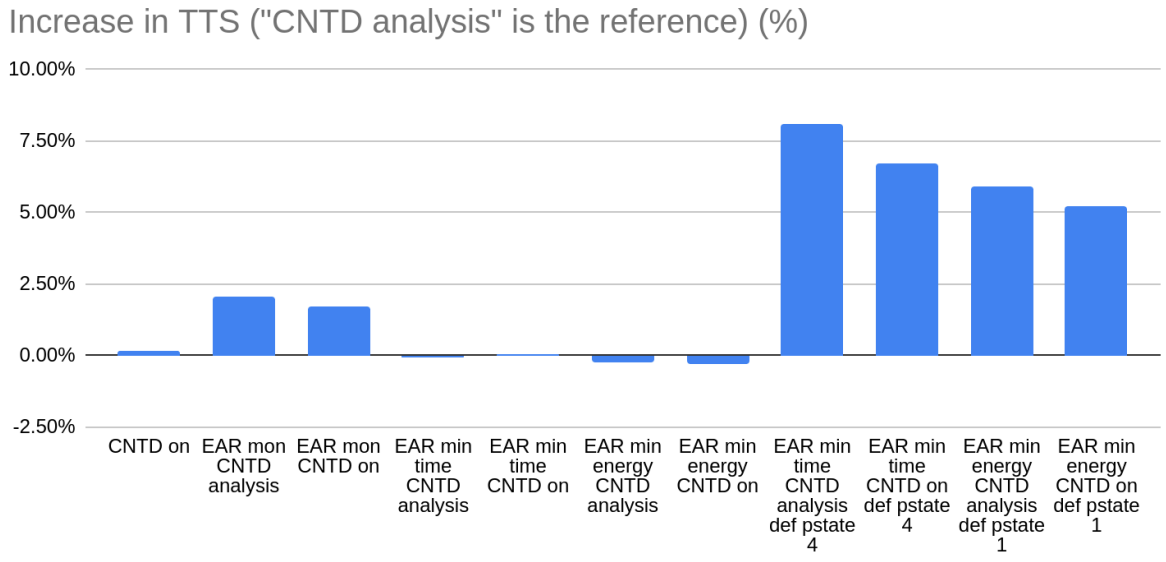
COUNTDOWN applied, in analysis mode; 2) CNTD on = COUNTDOWN applied, enabling the frequency reduction; 3) EAR mon = EAR using the policy monitoring; 4) EAR min time = EAR using the policy min_time; 5) EAR min energy = EAR using the policy min_energy; 6) def pstate 4 = EAR used with its default pstate (the fifth found on the list of the available frequency on the system, being the index starting from 0) for the policy min_time. If this specification is not present, then is the case where we explicitly modified it, setting it equal to the turbo pstate; 7) def pstate 1 = EAR used with its default pstate (the second found on the list of the available frequency on the system) for the policy min_energy. If this specification is not present, then is the case where we explicitly modified it, setting it equal to the turbo pstate.

Figure 4.1 shows the average frequency, the time spent reaching the final solution, the energy and power consumed at package level, plus the increases or reductions of the KPIs involved.




Power PKG (W)
1500
1000
500
0
CNTD analysis
CNTD on
EAR mon CNTD analysis
EAR mon CNTD on
EAR min time CNTD analysis
EAR min time CNTD on
EAR min energy CNTD analysis
EAR min energy CNTD on
EAR min time CNTD analysis def pstate 4
EAR min time CNTD on def pstate 4
EAR min energy CNTD analysis def pstate 1
EAR min energy CNTD on def pstate 1


Energy PKG (J)
800000
600000
400000
200000
0
CNTD analysis
CNTD on
EAR mon CNTD analysis
EAR mon CNTD on
EAR min time CNTD analysis
EAR min time CNTD on
EAR min energy CNTD analysis
EAR min energy CNTD on
EAR min time CNTD analysis def pstate 4
EAR min time CNTD on def pstate 4
EAR min energy CNTD analysis def pstate 1
EAR min energy CNTD on def pstate 1


Increase in TTS ("CNTD analysis" is the reference) (%)
10.00%
7.50%
5.00%
2.50%
0.00%
-2.50%
CNTD on
EAR mon CNTD analysis
EAR mon CNTD on
EAR min time CNTD analysis
EAR min time CNTD on
EAR min energy CNTD analysis
EAR min energy CNTD on
EAR min time CNTD analysis def pstate 4
EAR min time CNTD on def pstate 4
EAR min energy CNTD analysis def pstate 1
EAR min energy CNTD on def pstate 1

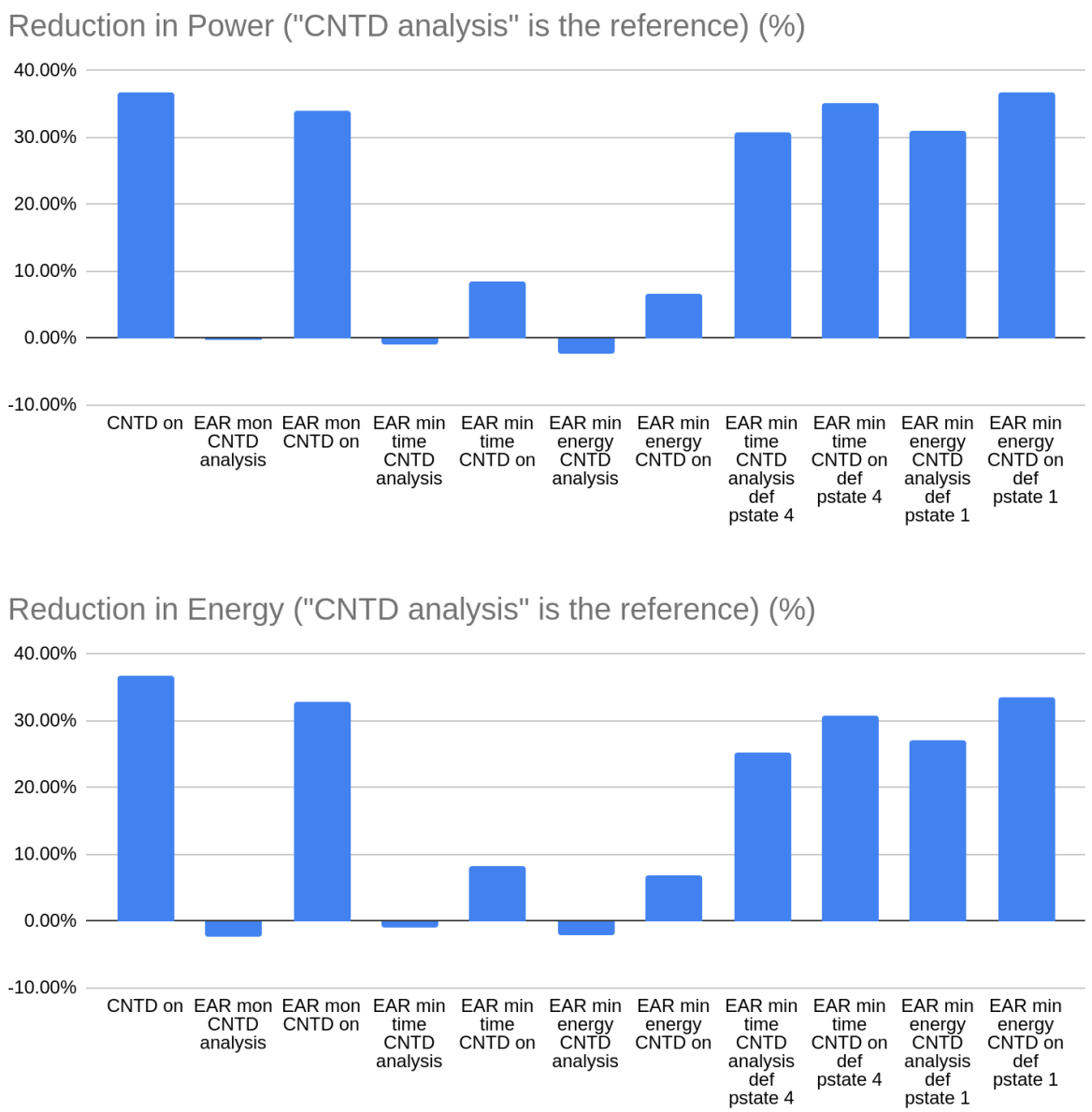


Figure 4.1: Average frequency, TTS, energy and power.

As we can see, the best case reported for reduced energy-to-solution KPI SO1.4 is the one in which only COUNTDOWN is involved. But that is explainable considering the fact that, at the moment, it just works with the maximum and minimum achievable frequencies, not considering any possible power management logic behind the resource utilization of the nodes involved.

This case is immediately followed by the one in which EAR is used with the *min_energy* policy enabled, in conjunction with the work done by COUNTDOWN, entering and exiting the MPI phases. In this last case, we in fact obtain a drastic reduction in power and energy consumption (around, respectively the 36% and 33%), by paying an increase in the time spent to reach the end of the benchmark, around the 5%. And spending around half a minute more, on a scale of 9 minutes, is not too much, compared with a power consumption reduced by more than one third.

It is worth noting that the configuration “EAR min energy CNTD on ” leads to a benefit both in terms of time-to-solution and energy-to-solution addresses at the same time SO1.2 and SO1.4 KPIs which is very promising.

4.3 Application-aware power capping with scheduler support

This integration scenario covers a use case where the job scheduler plays an active role in the power management of the system while the System Power Manager applies the cluster powercap, dynamically setting the node powercap based on application characteristics. The main goal of this scenario is to evaluate the performance of dynamic cluster powercap distribution with scheduler support against static powercap distribution (ie, every node has a certain amount of power budget at all times). More information about the internals of this scenario can be found in Deliverable 3.3, in its corresponding section. This deliverable pertains only to the evaluation of the results.

To evaluate this, we have simulated two supercomputing systems (a small one with 512 nodes and a large one with 10000 nodes) evaluating the performance of static powercap, dynamic powercap, and dynamic powercap with minimal scheduler support. Each platform has been evaluated with a workload generated using Lublin’s[2] model. Both cases simulate an homogeneous cluster with dual Intel Xeon Platinum 8480+ processors, with a TDP of 350W each, and with the DRAM portion consuming up to 75W per CPU. Since the main areas of power control that we have are CPU + DRAM (also known as package power, or PKG), we have used those metrics as the power indicators. This results in an average idle power of ~150-170W and a maximum of 850W. The power and speed curves of the processors have been based on experimental data obtained on real hardware while running several benchmarks and averaged out.
The scheduler support at this point is a simple communication that the cluster is running close to the maximum power budget (90% of the total power, in this case) so it can stop the execution of new jobs until power is freed. No jobs are canceled by this system, and no other changes to the scheduling are done.

Key performance indicators

The performance indicators are:

- Average job run time (time to solution), associated with SO1.3
- Average job wait time, associated with SO1.2
- Average job response time (derived from the previous two)
- Total energy consumed by the workload (including the energy consumed by nodes not currently running an active job), associated with SO1.4.

Small scale evaluation

For our small scale evaluation, we simulated a cluster of 512 nodes with a workload of 1000 jobs. We have calculated that the cluster, when running at it’s full capabilities should consume 435200 Watts (850W * 512 nodes) according to the TDPs of the CPUs and DRAM.

We have contemplated three scenarios for our simulations:

- 95% power budget: total cluster power must never exceed 413440 Watts
- 90% power budget: total cluster power must never exceed 391680 Watts

[2] https://www.cs.huji.ac.il/labs/parallel/workload/models.html#lublin99

- 85% power budget: total cluster power must never exceed 369920 Watts

For each scenario, the workload remains the same, with the only variation being the power allocation. We run each scenario three times: first with the baseline (with no power management from the System Power Manager) which has a fixed power allocation per node, then with the System Power Manager which distributes the power amongst the nodes according to their needs, and finally with the System Power Manager contacting the Job Scheduler when the cluster is at almost its limit to stop new job execution until power frees up.

Figure 4.2 shows the improvements to both time to solution and wait time for every scenario. The gains increase inversely to the power budget due to the job needs as well as the effects of the power distribution. For example, in the first case (95% power budget) an important percentage of jobs (>50%) can run at their desired settings without any performance penalty, due to their required power to run being lower than the power allocation of the nodes. As the power budget goes lower, the static powercap solution goes down in effectiveness since more and more jobs become power constrained. On the other hand, the dynamic solution has a small overhead in power allocation at the beginning of the jobs, but can usually run at the desired settings unless the cluster is close to full capacity, even in the more constrained scenarios.

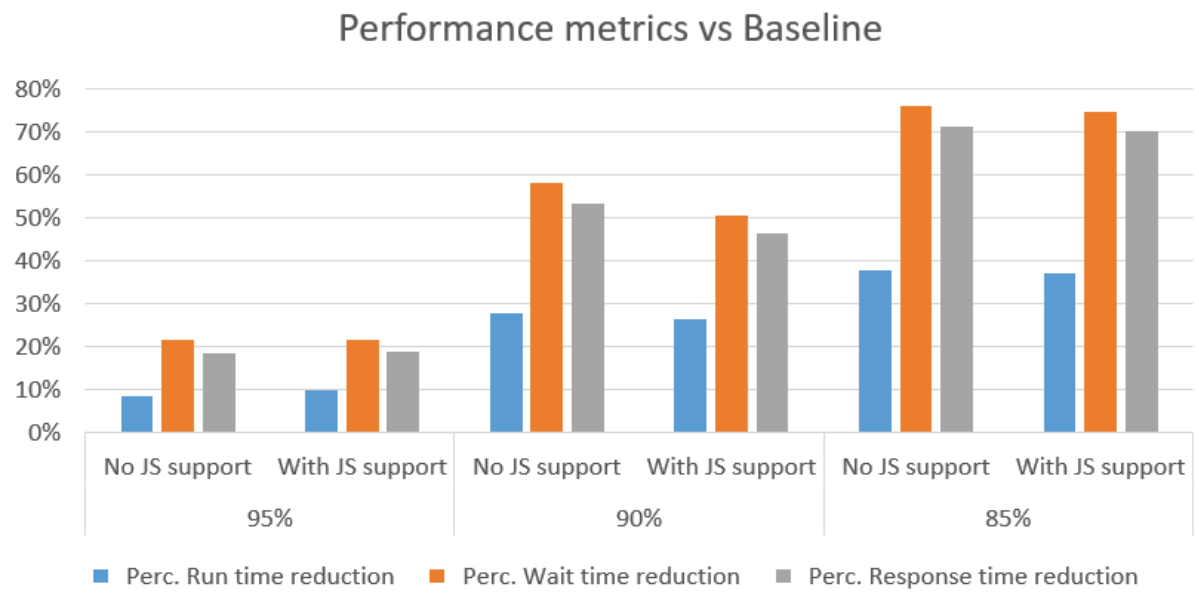


Figure 4.2: Reduction in average time to solution and average wait time for jobs compared to the baseline

In terms of energy efficiency, as can be seen in Figure 4.3 this scenario represents a substantial reduction of the energy consumed, especially as the power constraints go higher. This is directly related to the lower time to solution, since finishing the jobs early means freeing up the resources and either directly placing another job or reducing the power.

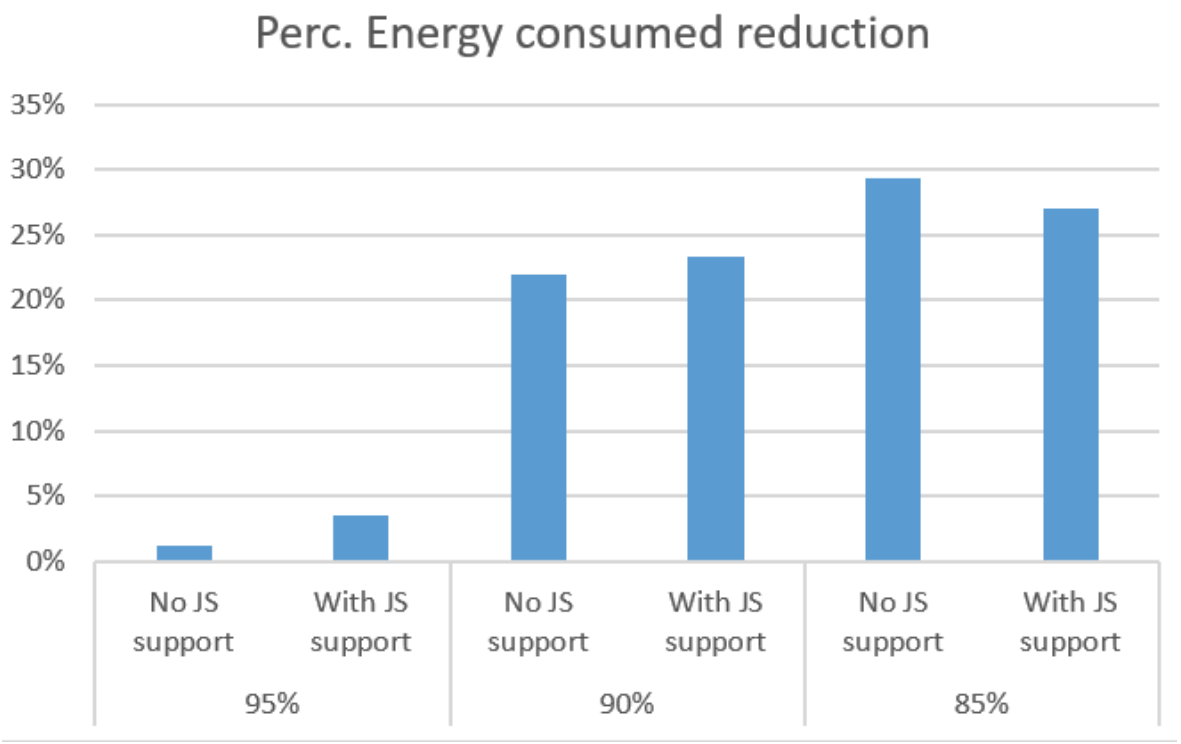


Figure 4.3: Energy savings compared to the baseline

Large scale evaluation

For our large scale evaluation, we simulated a cluster of 10000 nodes with a workload of 20000 jobs. Just like with the small scale evaluation, we have calculated that the cluster, when running at it's full capabilities should consume 435200 Watts (850W * 512 nodes) according to the TDPs of the CPUs and DRAM.

We have contemplated the same three scenarios for our simulations:

- 95% power budget: total cluster power must never exceed 8075 KWatts
- 90% power budget: total cluster power must never exceed 7650 KWatts
- 85% power budget: total cluster power must never exceed 7225 KWatts

We have used the same methodology as before, running each scenario with static powercap, dynamic powercap (SPM-only) and dynamic powercap with job scheduler support (SPM+JS).

Figure 4.4 shows similar results as seen on the small scale simulation, with moderate gains on the 95% scenario and substantial ones in 90% and 85%. In this case, the generated trace differs slightly from the previous one, with the cluster reaching full saturation (ie, having jobs on most or all nodes concurrently) less frequently, as well as having jobs slightly more spaced out. This results in smaller gains comparatively, but still performing above the baseline in all cases.

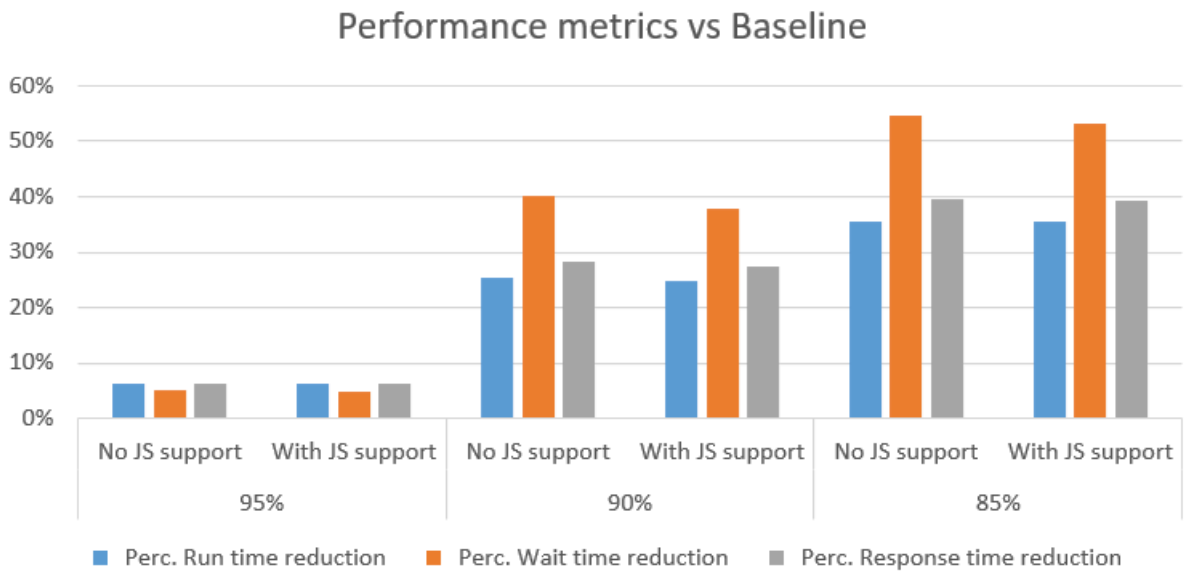


Figure 4.4: Reduction in average time to solution and average wait time for jobs compared to the baseline

In regards to energy efficiency, the spacing of the jobs means that running at non-target settings can sometimes result in better time/energy efficiency. This is especially notable in the 95% power budget case, (as can be seen in Figure 4.5) where running at the 2nd highest frequency (which the static powercap does) is slightly more efficient than running at target settings. Nonetheless, as we reach higher power constraints the solution quickly becomes more efficient in terms of both time and energy.

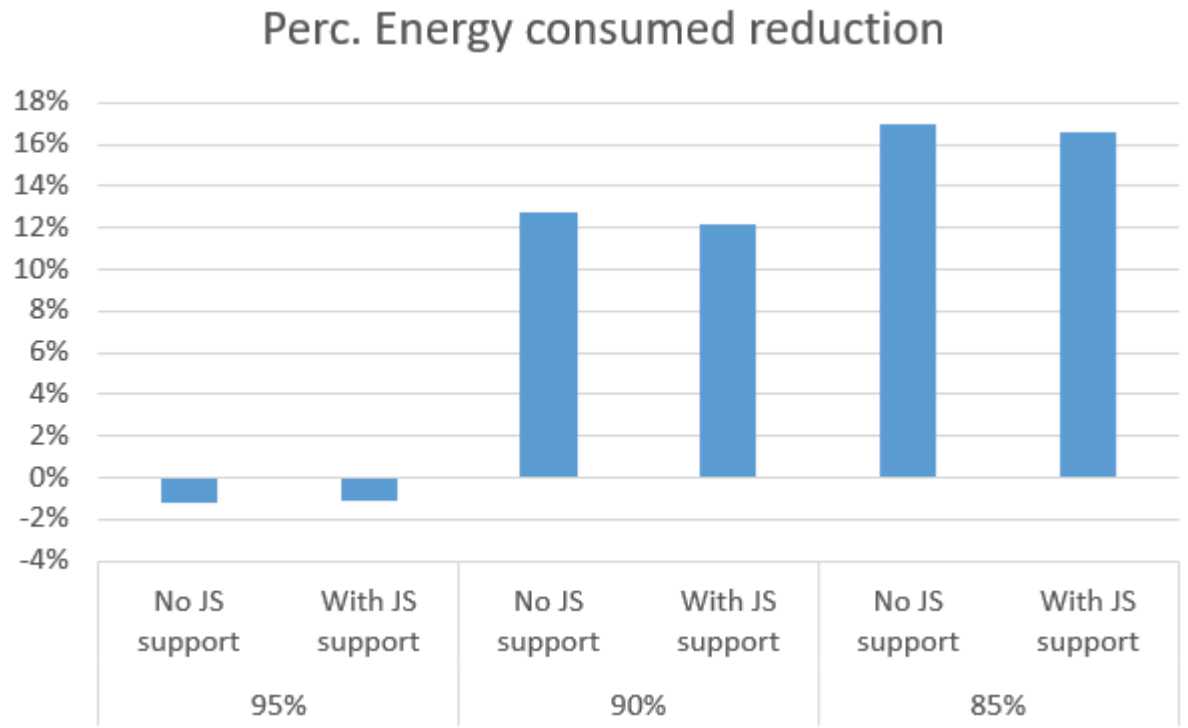


Figure 4.5: Reduction in the energy consumption of the trace compared to the baseline

In regards to the objectives set in Table 3.1, with the exception of the 95% scenario where the constraints may be a bit too lax for our proposal, we have met the projected improvements (SO1.3). In terms of speed increase and throughput we have an increase of over 20% in application speed versus the baseline, as well as over a 30% reduction in average job wait time. Similarly, in terms of energy efficiency we get a reduction of energy consumption over the projected 10%. In addition to that, we can see that the solution scales relatively well from a small-medium cluster (~500 nodes) to a large one (10000 nodes), especially in the more constrained cases. Finally, the solution seems to improve versus the baseline the higher the constraints, which brings interesting avenues to explore in further work as can be seen in the next section.

Job scheduler support as a fallback and future improvements

As can be seen in the previous evaluation, having the job scheduler stop new job executions as we reach a higher amount of power is not usually the optimal solution when running under normal constraints, where the simple dynamic allocation is better.

That said, in situations where the power constraints go even lower (75% or below), the Job Scheduler stopping new job executions results in both better run times and wait times as well as energy efficiency.

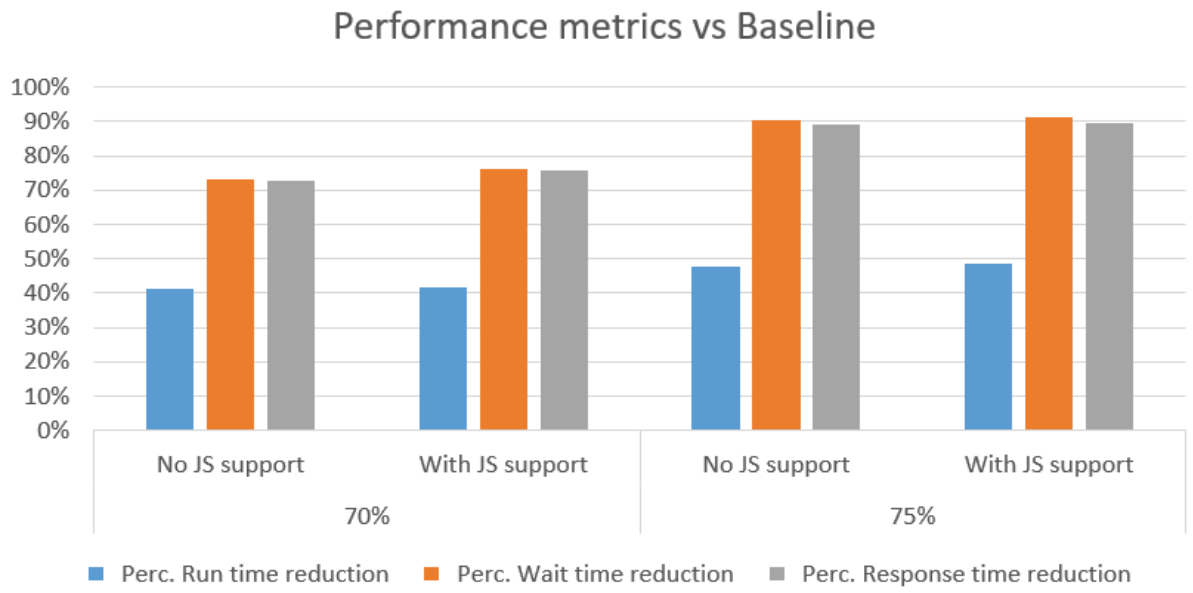


Figure 4.6: changes in average time to solution and average wait time for jobs compared to the baseline in a heavily constrained environment

Furthermore, as we go lower in power budget there is no feasible way to keep all the nodes running with a constrained power without having to shut most of them down, as there is only so much power limiting that can be done while a node is running. This can be seen in Figures 4.6 and 4.7 where both the performance and energy metrics exceed the previous cases, partially due to the inability of static powercap to handle such situations.

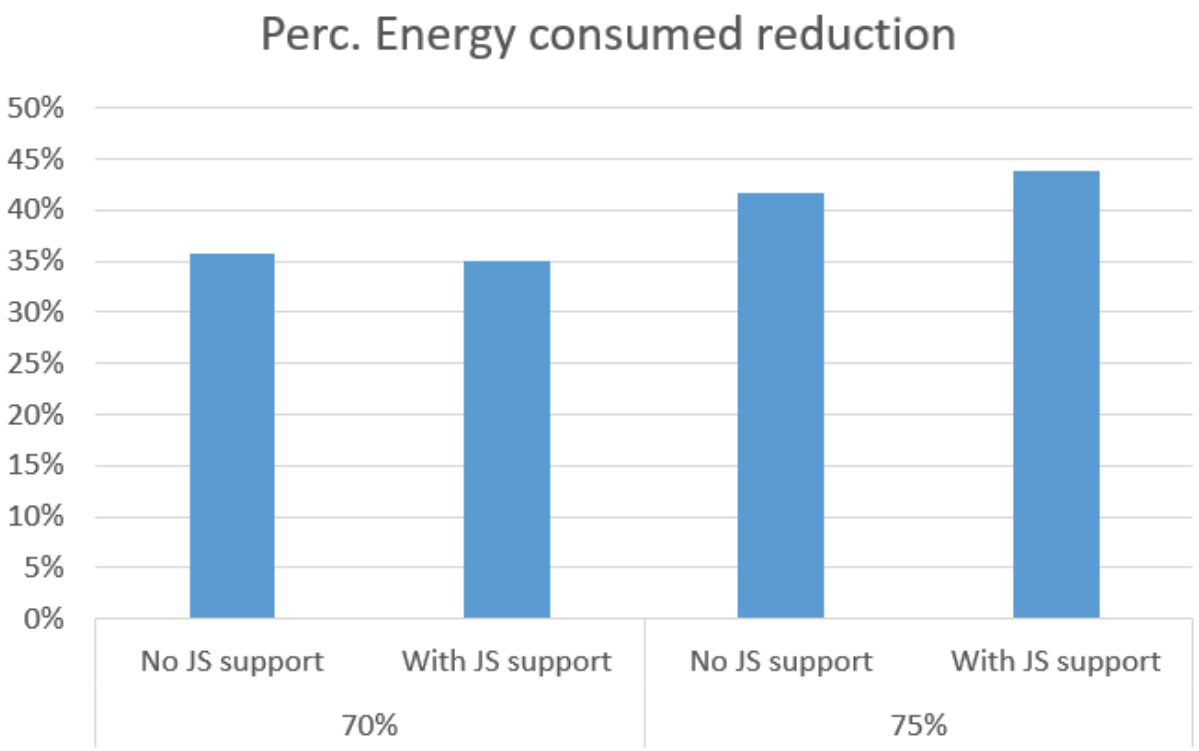


Figure 4.7: changes in the energy consumption of the trace compared to the baseline in a heavily constrained environment

This can be particularly useful in situations where the power settings have to change abruptly and harshly due to external circumstances. For example, in cases where there has been a power grid failure that results in lowered available total power, the Job Scheduler support can help by not having to manually shut down several nodes and instead having the SPM-JS combination automatically lower the system's consumption while keeping throughput at a maximum.

Finally, as a next step a dynamic switch between using the Job Scheduler and not depending on the power constraints would be feasible, and improve the overall solution in both cases with low constraints as well as highly constrained systems while also supporting a dynamic environment where the constraints may change due to external circumstances. Furthermore, a next integration that does not directly depend on power consumption but on a related metric would be easy to implement. An example would be systems where, especially in summer, the cooling system cannot keep up with the cluster running at full power. In this case, using the temperature as an indicator the System Power Manager could tell the Job Scheduler to stop new jobs from starting and thus prevent new sources of heat until the system cooled off.

5. REGALE Library

The REGALE library aims to effectively materialize the PowerStack initiative within the REGALE project, providing a single layer of communication among all the tools, libraries and software, composing this big ecosystem. This framework is based on the DDS[3] and RTPS[4] protocols and on FastDDS as their implementation, and to the best of our knowledge it is the first of its genre, enabling all the different entities involved in the REGALE ecosystem (but not limited only to these) to communicate and exchange messages (related to information or even to possible commands), without any further modification inside their internals. In this document, we report some initial proof of concept, functionality tests and some initial evaluation tests made on the DDS implementation currently used in the REGALE library (FastDDS).

5.1 Functionality tests

The benchmark used for the proof of concept of the REGALE library is a simple MPI program performing an *MPI_Alltoall*, followed by some computations, to speed up again the frequency at the end of the MPI phase. We controlled the actual exchange of communication messages among the involved tools to take specific actions related to power and energy efficiency and monitoring.

The test was made on the E4 infrastructure, using three different nodes each hosting one different entity: a Job Manager, a Node Manager, and a Monitoring System, in order to check the feasibility of exchanging messages among different nodes, using different transport protocols than the *shared memory* one (intra-node communication); in this specific case, we used *UDP*.

Regarding the Job Manager, the COUNTDOWN implementation has been considered, while for the other two entities we have used the synthetic components we developed as part of the REGALE library repository, which act as possible substitutes (very simple ones) of other and more complicated implementations of the same components. Plus, these synthetic components can be built with the REGALE library from possible users, to allow them to better understand the usability and the capabilities of this new layer. We decided to use them for the sake of simplicity, since the result does not change when plugging them off and inserting at their places more mature implementations like EAR and EXAMON: we underline again the fact that we just need to call the same APIs, no matter what implementations are communicating. They can be, in fact, easily interchanged (as reported in the deliverable D3.3) with EAR and EXAMON.

Specifically, the synthetic monitoring was programmed to send a reply to the requests, with the highest frequency possible being 2GHz. Figure 5.1 shows the outcome after launching

[3] The DDS specification describes a Data-Centric Publish-Subscribe (DCPS) model for distributed application communication and integration. This specification defines both the Application Interfaces (APIs) and the Communication Semantics (behavior and quality of service) that enable the efficient delivery of information from information producers to matching consumers. The purpose of the DDS specification can be summarized as enabling the 'Efficient and Robust Delivery of the Right Information to the Right Place at the Right Time.' https://www.omg.org/spec/DDS/About-DDS/

[4] This specification defines an interoperability wire protocol for DDS. Its purpose and scope is to ensure that applications based on different vendors' implementations of DDS can interoperate. https://www.omg.org/spec/DDSI-RTPS/2.2

the benchmark using 16 MPI processes. Figure 5.2 shows the breakdown of time in computation (APP time) and communication (MPI time).

```
AVG CPU frequency:                              1456 MHz
```

Figure 5.1: Final average frequency

```
##################### MPI TIMING #####################
APP time: 234.338 sec (52.76%)
MPI time: 209.797 sec (47.24%)
TOT time: 444.135 sec (100.00%)
```

Figure 5.2: Total time of the run and its specific percentage 1) into MPI phases, and 2) into APP phases (not MPI ones)

We observe that the average frequency reported by COUNTDOWN was 1.4GHz; with simple calculations, we can obtain the same value from the percentage of time spent in MPI and APP events, and from knowing the maximum available frequency (now 2GHz) and the minimum one (still 800MHz):

$$(0.5276*2GHz) + (0.4724*0.8GHz) = 1.4GHz$$

Moreover, analyzing the content of the files *scaling_max_freq* and *scaling_min_freq*, we can see that the right number of frequencies have been changed (Figure 5.3).

```
[ftesser@iwnode09 ~]$ cat /sys/devices/system/cpu/cpu*/cpufreq/scaling_max_freq | grep "2000000" | wc -l
16
[ftesser@iwnode09 ~]$ cat /sys/devices/system/cpu/cpu*/cpufreq/scaling_min_freq | grep "800000" | wc -l
16
[ftesser@iwnode09 ~]$
```

Figure 5.3: The *scaling_max_freq* and *scaling_min_freq* files are the ones modified by COUNTDOWN to set the actual running frequencies entering and exiting MPI phases. As we can see, at the end of the benchmark, run on 16 MPI processes, the frequencies were corrected modified for the cores pinned by the simulation and that the synthetic Monitor system reports the correct frequencies (Figures 5.4 and 5.5).

```
/0 node_id 0 metric 5 value 800.030151 start_time 1713448308 end_time 1713448308
/1 node_id 0 metric 5 value 799.991638 start_time 1713448308 end_time 1713448308
/2 node_id 0 metric 5 value 800.023010 start_time 1713448308 end_time 1713448308
/3 node_id 0 metric 5 value 799.995728 start_time 1713448308 end_time 1713448308
/4 node_id 0 metric 5 value 800.037354 start_time 1713448308 end_time 1713448308
/5 node_id 0 metric 5 value 799.974121 start_time 1713448308 end_time 1713448308
/6 node_id 0 metric 5 value 799.978882 start_time 1713448308 end_time 1713448308
/7 node_id 0 metric 5 value 800.019104 start_time 1713448308 end_time 1713448308
/8 node_id 0 metric 5 value 800.006836 start_time 1713448308 end_time 1713448308
/9 node_id 0 metric 5 value 800.044922 start_time 1713448308 end_time 1713448308
/10 node_id 0 metric 5 value 800.007812 start_time 1713448308 end_time 1713448308
/11 node_id 0 metric 5 value 799.989929 start_time 1713448308 end_time 1713448308
/12 node_id 0 metric 5 value 799.993042 start_time 1713448308 end_time 1713448308
/13 node_id 0 metric 5 value 800.066528 start_time 1713448308 end_time 1713448308
/14 node_id 0 metric 5 value 799.955017 start_time 1713448308 end_time 1713448308
/15 node_id 0 metric 5 value 800.053162 start_time 1713448308 end_time 1713448308
```

Figure 5.4: The synthetic Monitoring system is reporting the messages sent by the J.M., regarding the actual frequency inside the MPI_Alltoall (the minimum one feasible by the system, 800MHz)

```
/0 node_id 0 metric 5 value 1976.697144 start_time 1713448310 end_time 1713448310
/1 node_id 0 metric 5 value 1999.987305 start_time 1713448310 end_time 1713448310
/2 node_id 0 metric 5 value 1999.992920 start_time 1713448310 end_time 1713448310
/3 node_id 0 metric 5 value 1999.992432 start_time 1713448310 end_time 1713448310
/4 node_id 0 metric 5 value 1999.992798 start_time 1713448310 end_time 1713448310
/5 node_id 0 metric 5 value 1999.997803 start_time 1713448310 end_time 1713448310
/6 node_id 0 metric 5 value 1990.150757 start_time 1713448310 end_time 1713448310
/7 node_id 0 metric 5 value 1985.090698 start_time 1713448310 end_time 1713448310
/8 node_id 0 metric 5 value 1981.760376 start_time 1713448310 end_time 1713448310
/9 node_id 0 metric 5 value 1996.507935 start_time 1713448310 end_time 1713448310
/10 node_id 0 metric 5 value 1999.991333 start_time 1713448310 end_time 1713448310
/11 node_id 0 metric 5 value 1999.996460 start_time 1713448310 end_time 1713448310
/12 node_id 0 metric 5 value 1999.994385 start_time 1713448310 end_time 1713448310
/13 node_id 0 metric 5 value 1999.991821 start_time 1713448310 end_time 1713448310
/14 node_id 0 metric 5 value 1983.101074 start_time 1713448310 end_time 1713448310
/15 node_id 0 metric 5 value 1985.778442 start_time 1713448310 end_time 1713448310
```

Figure 5.5: The synthetic Monitoring system is reporting the messages sent by the J.M., regarding the actual frequency exiting the MPI_Alltoall (reaching the maximum frequency set by the synthetic N.M., which was 2GHz)

Overall, we show that the core functionality is in place, and, more importantly, that we needed less modifications in the implementations of the actors in play, compared, for example, with what was needed in the REGALE prototypes (integration scenarios). Here in fact, we just added a couple of APIs in the Job Manager to let it work and communicate with a Node Manager and a Monitoring. Specifically, the calls involved were:

1) *regale_monitor_init* (to initialize the Monitor client)
2) *regale_nm_init* (to initialize the Node Manager client)
3) *regale_report_job_telemetry* (to send current frequency to the Monitoring system)
4) *regale_nm_get_current_conf* (to ask to the Node Manager the actual maximum freq)
5) *regale_monitor_finalize* (to finalize the Monitor client)
6) *regale_nm_finalize* (to finalize the Node Manager client)

We obtained the previous results without taking into account all the things that were considered in the first version of the Integration Scenario, where we were required to take under control more things, among which: 1) the right usage of the correct governor (*Userspace*), 2) the multiple, parallel and concurrent accesses to the files *scaling_setspeed*, 3) the already cited problem (the deliverable D3.2 contains more detailed information and explanations regarding the just mentioned criticalities) of the Job Manager, to not been able to recognize if the value equal to the minimum one (contained in the *cpuinfo_min_freq* file) has been something decided by the Node Manager, or something previously written by the Job Manager itself.

To conclude, as we previously said, we report here some graphs related to the results obtained from the evaluation tests we performed on the FastDDS implementation, considering that something quite important, being this specific implementation the cornerstone of the REGALE library, concerning the DDS overall system.

5.2 Preliminary evaluation

Our second set of experiments was performed on 10 nodes of the production system G100, at CINECA. Specifically, the nodes include 2 x CPU Intel CascadeLake 8260, each. Plus, the transport used were UDP and UDPM (UDP Multicast).

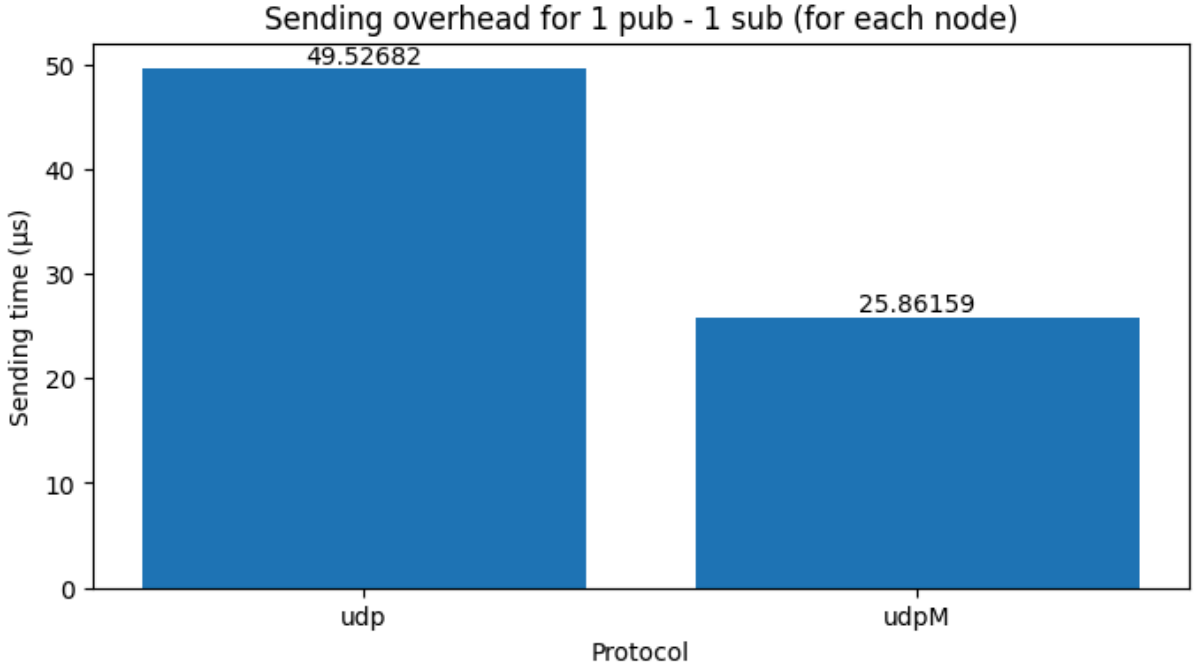


Figure 5.6: Sending times with one subscriber per node (10 node used)

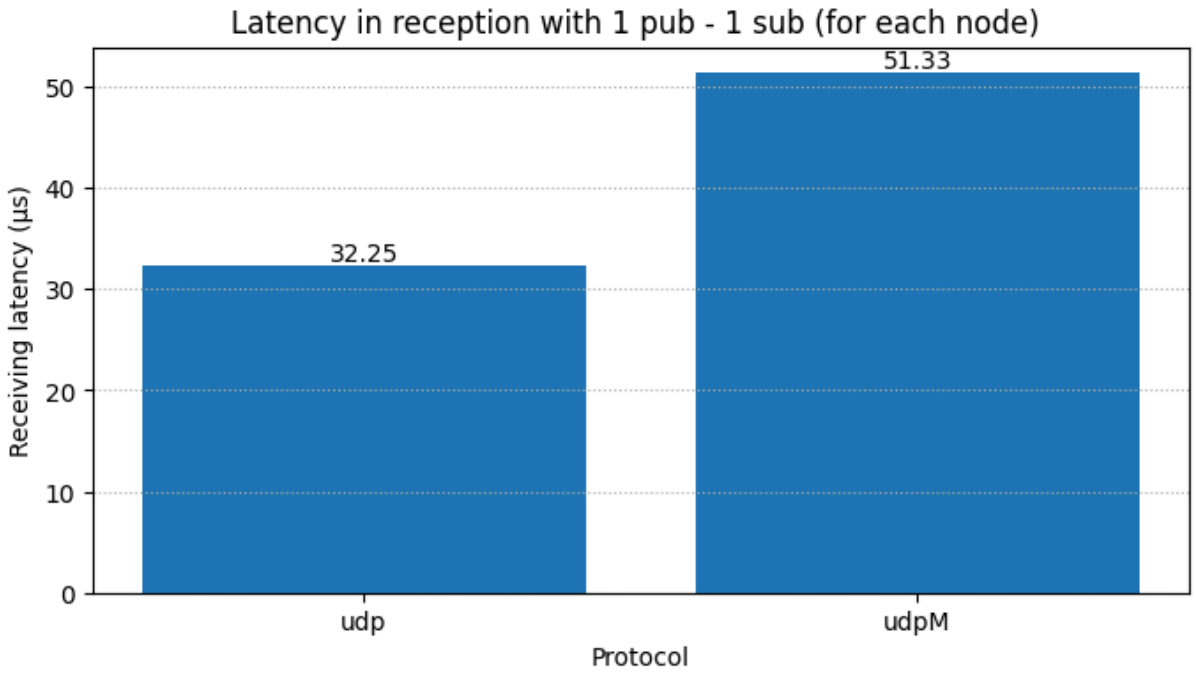


Figure 5.7: Latencies with one subscriber per node (10 used)

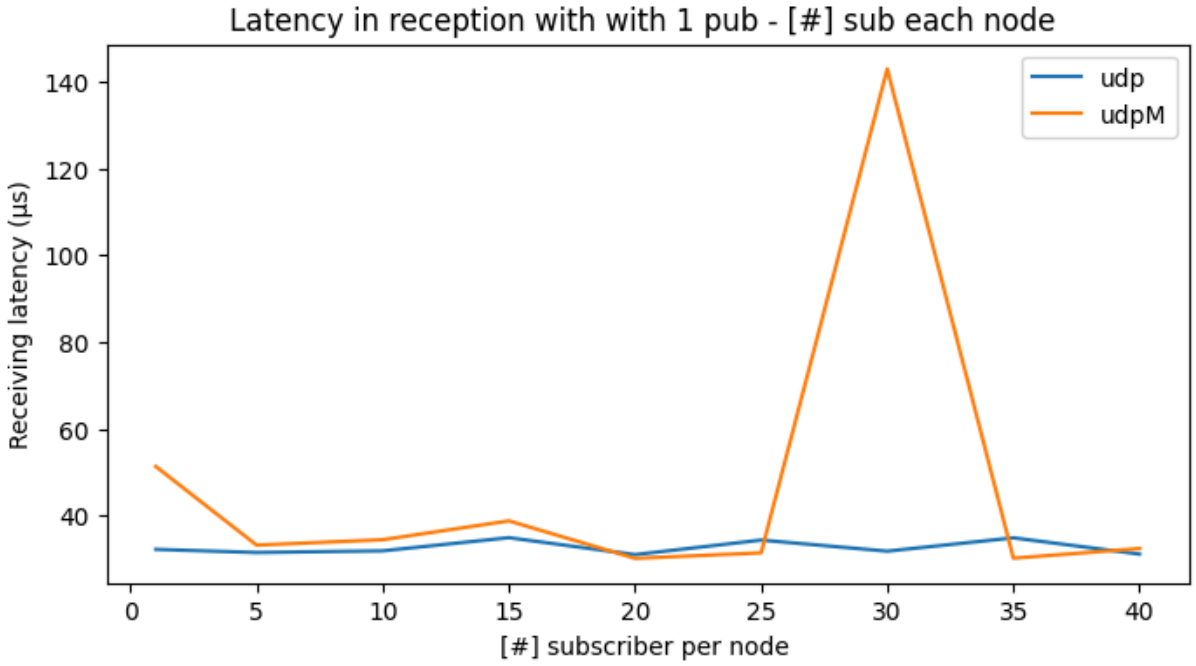


Figure 5.8: Latencies with the number of subscriber varying for each node (10 used)

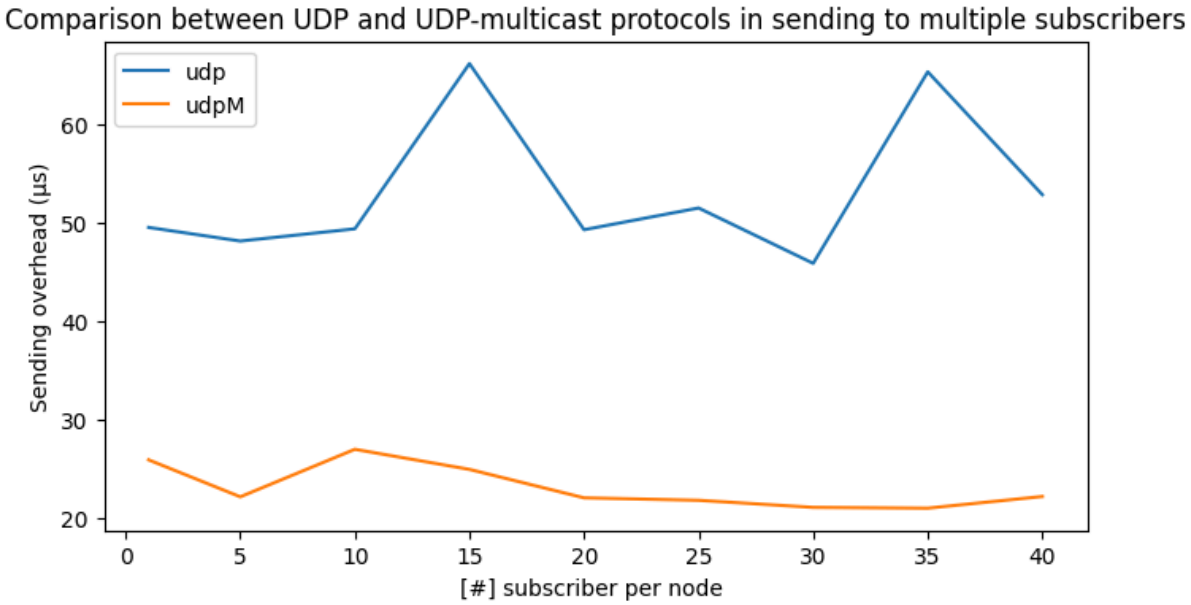


Figure 5.9: Here are reported the sending overheads with the number of subscriber varying for each node (10 used)

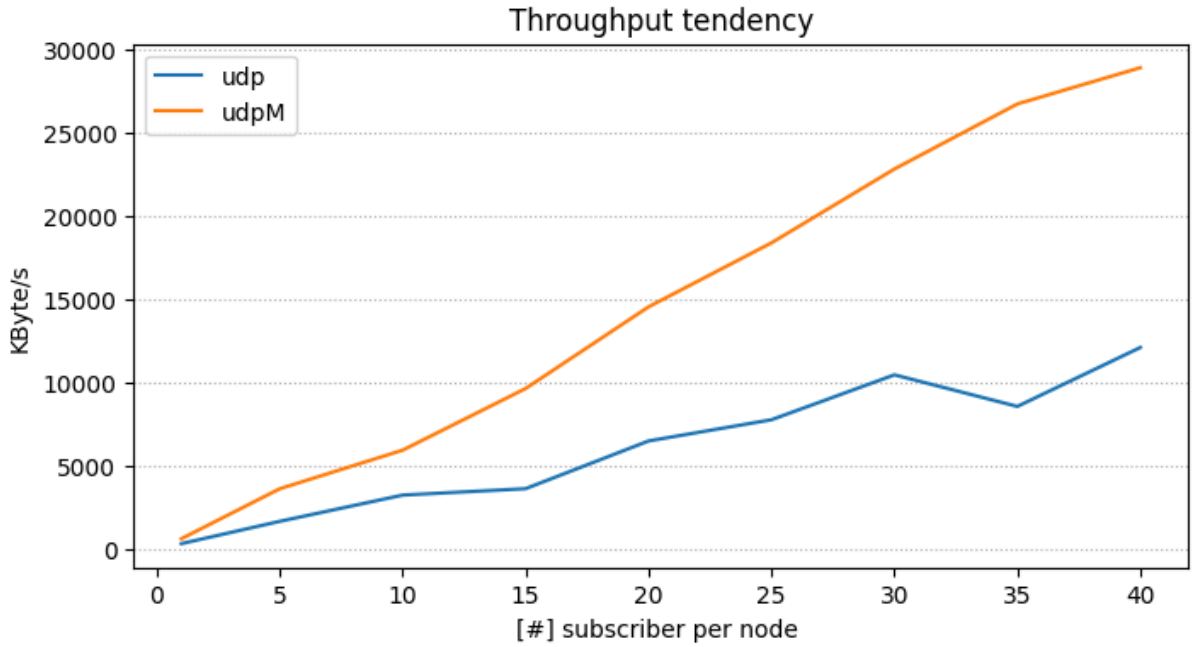


Figure 5.10: Throughput is reported here, with the number of subscriber varying for each node (10 used)

As we can see, in Figures 5.6-5.10, the latencies and the overheads required, are in line with the time needed by the hardware to adopt the modifications for the usage of the p-states (more or less around 500 microseconds, on an x86_64 architecture).

6. REGALE pilots

6.1 Pilot 1: Industrial Scale Unsteady Adjoint-based Shape Optimization of Hydraulic Turbines

State-of-practice: Pilot 1 is dealing with the design/shape optimization of a hydraulic turbine for minimum pressure pulsations of the flow through the turbine. So far, industries were dealing with this problem through a trial-and-error procedure, which has an increased cost (high turn-around time) and may lead to suboptimal solutions. On the other side, academia has intensively been working on the development of efficient optimization methods to tackle real-world applications at reasonable wall-clock times.

During REGALE, the optimization problem of Pilot 1 was carried out using evolutionary algorithms (EAs) assisted by a) surrogate evaluation models (or metamodels; MAEA) to avoid "useless" calls to the computationally expensive CFD evaluation s/w and b) the Principal Component Analysis (PCA) to tackle the “curse of dimensionality”, i.e. the performance degradation of EAs in problems with many design variables. The use of Deep Neural Networks (DNNs) as surrogates during the optimization has also been exploited. Thus, optimal designs, for the selected runner blade shape parameterization, were obtained at reasonable computational cost.

Irrespective of the shape optimization, the performance metrics of hydraulic turbines are highly affected by the imposed boundary conditions and "feel" uncertainties in the flow rate, the inlet flow angle etc. To quantify these uncertainties (Uncertainty Quantification; UQ) methods such as the non-intrusive variant of the Polynomial Chaos Expansion (niPCE) are used. These require a series of flow analyses at different boundary conditions, the results of which are stored and, then, post-processed to compute the statistical moments of the quantities of interest, for instance the mean value and standard deviation of the efficiency, the head, etc. In REGALE, such UQ studies were performed through the Melissa workflow manager using a storage-free approach and with the ‘optimal’ allocation of computational resources for concurrent simulations.

Over and above the optimization and UQ methods, the CFD s/w (PCOpt/NTUA in-house code PUMA) used for the analyses is GPU-accelerated and contributes to the reduction of the computational cost of each simulation.

REGALE value proposition: During REGALE, the shape optimization of the runner of a hydraulic turbine was performed using both the standard EA and the PCA-driven MAEA and comparisons in terms of the optimization turnaround time and/or the obtained optimized solution(s) for a given computational budget were made. The optimization runs were performed on a computational node with 4 NVIDIA A100 GPUs and the search was parallelized, i.e. concurrent evaluation of four candidate solutions. DNNs were also used as surrogates to the mixing plane technique, reducing the number of domains that need to be simulated during each analysis and thus the computational cost per evaluation. The obtained gain in optimization turnaround time was in accordance with SO1.1 (Improved application performance); the use of surrogates reduced by 50% (or speed-up 2x) the computational cost required to reach a solution of the same quality to that of the standard EA. The detailed comparisons and plots are included in D4.3. Regarding UQ studies, comparisons on the required storage to compute the statistical moments with and without the use of Melissa were carried out.

Evaluation scenarios: The activities are expected to reduce the requirements in a) programming for 'managing' (assigning to the available resources) the various simulations required for UQ studies and the post-processing of the so-stored results and b) storage of the flow fields. This is extremely beneficial, particularly in large scale applications. For the latter, it is important to ensure the good scalability of the PUMA s/w to many-GPU systems. This will allow the simulation of larger computational domains and, overall, make PUMA ready for exascale deployment. Given the above, the following evaluation scenarios were tested:

- Scenario 1: Coupling of PUMA with the Melissa workflow manager for UQ studies.
  - Baseline: Customized scripts for orchestrating the computations required and full storage of the results files in order to be post-processed for UQ.
  - KPI - Storage Requirements: Evaluate the reduction in the storage requirements when Melissa is used for UQ analyses.
  - KPI - Programmability: User and Developer Experience: Evaluate the ease of coupling PUMA with Melissa.

- Scenario 2: PUMA s/w scalability tests on multi-GPU systems
  - Baseline: PUMA s/w GPU deployment optimized for single GPU simulations.
  - KPI - Application speedup: Evaluate the speed-up by increasing the workload and the number of GPUs.

Evaluation results:
Scenario 1: The first evaluation scenario is related to the coupling of PUMA with the Melissa workflow manager for UQ studies. The so-developed interface is described in D4.3. The reduction in storage requirements depends on the quantity of interest (QoI) and the data that need to be stored from each CFD analysis and post-processed in order to compute the statistical moments of the QoI. Table 6.1 summarizes the sizes of the basic files (related to the computation of the quantities of interest) stored by PUMA when simulating the flow on a relatively coarse mesh (that of Pilot 1) as well as a fine mesh from another hydraulic turbine geometry. Assuming that M simulations need to be performed for UQ, using Melissa for computing the statistical moments reduces the required storage from M to 1/M. In the UQ problems considered in Pilot 1 (with one or two uncertain variables), the value of M varies from 3 to 16; thus the reduction in storage may be from a few MB to dozens of GB.

Table 6.1: Size of basic files stored by PUMA.

Type of data	Binary file size (MB)	
	Coarse mesh	Fine mesh
Runner blade surface	1.6	2
Runner domain iso-stream	2.3	8.2
Full Storage	553	1609

As far as the programmability and developer/user experience are concerned, a) interfacing PUMA with Melissa requires minimum programming and/or scripting skills, thus can be easily accomplished even by non-experienced developers and b) the Melissa workflow manager undertakes the automatic allocation and use of the resources required for the numerous simulations. These are in contrast to the development of customized scripts/workflows to

orchestrate the 'optimal' use of the available resources, store any required data and post-process them.

To sum up, the above described gains from the UQ workflows using Melissa are in accordance both with the SO1 (Effective utilization of resources) and the SO3 (Easy and flexible use of supercomputing services).

Scenario 2: In this scenario the scalability of PUMA on multi-GPUs systems is examined. The speed-up is evaluated by increasing the number of GPUs used to simulate the flow in a given computational domain, i.e. the workload is practically specified by the number of nodes. Different sizes of computational grids from hydraulic turbines were tested. The simulations for this evaluation scenario were performed mainly on the GRNET SA high-performance computing system ARIS (Advanced Research Information System). Specifically, the ML node of ARIS with 2 x Intel(R) Xeon(R) E5-2698v4, 512GB RAM and 8 x NVIDIA V100 GPUs (16GB memory each) was used. The reason for using only this node of the ARIS HPC system was related to the drivers and compilers the PUMA s/w requires (NVIDIA driver 470.57.02 or higher, CUDA 11.4 etc). Three computational meshes with ~4.5Mi (coarse), ~12.5 Mi (medium) and ~35Mi (fine) nodes, respectively, were used and the results are summarized in Table 6.2.

Table 6.2: Summary of the results from evaluation scenario 2 on the ARIS HPC system.

Computational Grid	Coarse	Medium	Fine
Number of Nodes	4572823	12427323	35065403
# GPUs	**Seconds per iteration / Mean GPU memory usage (%)**		
1	1.71 / 85.7	-	-
2	0.97 / 52.4	-	-
3	0.76 / 41.1	2.08 / 97.7	-
4	0.68 / 35.5	1.61 / 78.3	-
6	-	1.17 / 58.4	1.43 / 97.2
8	-	0.95 / 33.8	1.26 / 65.8

The CFD simulation on the coarse mesh can be carried out on a single GPU by using 85.7% of its memory. A speed-up of 1.76x is achieved when using 2 GPUs. This mesh is not expected to scale well when increasing the number of subdomains and as a consequence the GPUs; a speed-up of 2.5x is achieved on 4 GPUs. For the medium sized mesh at least 3 GPUs are required for the simulation. A speed-up of 1.77x is obtained when duplicating the number of GPUs, i.e. from 3 to 6. This mesh scales well up to the 8 available GPUs. This is the case for the fine mesh which needs at least 6 GPUs for the flow simulation.

A mesh of ~45Mi nodes was the largest that could be simulated on the ARIS system. In order to check the scalability of PUMA on finer meshes a PCOpt/NTUA cluster node with 4 NVIDIA

A100 GPUs (3 with 40GB memory and one with 80GM memory) was additionally used. On this node the simulation of meshes from 40Mi to 103Mi nodes. Each iteration of the finest possible mesh that could run to this node (using ~99% of the available memory of all GPUs) took 0.87seconds. The above-mentioned tests are inline with SO2 (Broad applicability scalability). The restructuring of the parallel deployment and reduction in memory requirements of the PUMA s/w performed during REGALE allowed for the simulation of computational meshes of up to 100Mi nodes.

6.2 Pilot 2: In-Transit Workflow for Ubiquitous Sensitivity Scope: Very large scale Sensitivity Analysis Analysis and MetaModel Training. Application to Infrastructure Safety.

State-of-practice: Multiple simulation runs (sometimes several thousands) are required to perform uncertainty quantification studies, complex optimization, data assimilation or recently for training machine learning metamodels. Current practice consists in running all the necessary instances with different sets of input parameters, storing the results to disk, often called ensemble data, to later read them back from disk to compute the required data processing. The amount of storage needed may quickly become overwhelming, with the associated long read time that makes data processing time consuming. To avoid this pitfall, scientists reduce their study size by running low resolution simulations or down-sampling output data in space and time. Today terascale and tomorrow exascale machines offer compute capabilities that would enable large scale studies ranging from uncertainty quantification to training metamodels. But they are unfortunately not feasible due to this storage issue.

REGALE value proposition: Novel approaches are required. In situ and in transit processing emerged as a solution to perform data analysis starting as soon as the results are available in the memory of the simulation. The goal is to reduce the data to store to disk and to avoid the time penalty to write and then read back the raw data set as required by the classical postmortem analysis approach. To our knowledge the only available in transit solution for dealing with large scale multiple simulation runs is the open-source software Melissa. In the context of REGALE, UGA and EDF collaborated to run a multiple simulations study of an industrial case. EDF will provide the model and parameters for large-scale CFD simulations using OpenTelemac (www.opentelemac.org).

Evaluation scenario: The simulations have been run as a large ensemble and the produced data were processed online in two different workflows, using the Melissa framework developed by EDF and UGA:

- The first workflow will target a direct sensitivity analysis to generate various ubiquitous statistics, i.e. high-resolution spatio-temporal statistics fields, including advanced ones like Sobol indices and quantiles.
- The second workflow will first train a deep neural network metamodel on-line from the data produced by the simulations, still using Melissa.

Sensitivity Analysis

We aimed at performing a sensitivity analysis large run with Melissa. Before REGALE, Melissa was connected to a single Computational Fluid Dynamics (CFD) code, the open-source Code_Saturne (www.code-saturne.org). In this pilot, Melissa has also been connected to the open-source suite OpenTelemac (www.opentelemac.org).

A use case concerning river floodings was selected. The reason is that, as the global climate has been changing, the number and intensity of natural disasters have significantly increased in recent decades. Owing to the frequent occurrence of floods caused by heavy rain, urban floods or floods near power plants have become an important question, and research interest has increased. The selected use case models a section of the Garonne river in the south of France. This use case does not contain any urban areas or power plants, thus it can be shared with the scientific community. Flow depth estimation that is often limited by uncertainties in hydrodynamic numerical models. In order to overcome these limits, uncertainties should be analyzed. This use case investigates the effect of two uncertainty sources on the water level calculation:, the roughness coefficient and the upstream discharge. Indeed, the hydraulic roughness is uncertain because flow measures are not available or reliable for calibration and validation. Discharge is also uncertain because it results from extrapolation of discharge frequency curves at very low exceeding probabilities.

Tests were performed on the EDF's corporate supercomputer Cronos (https://www.top500.org/system/179899/). Melissa and Telemac were installed and used in this system.

During this study, Melissa Server treated on-the-fly the data coming from the simulations. We would like to remark that, in a classical study, all this data would be output to the filesystem, and read back to compute statistics, which is current practice for EDF simulation engineers. We observed that avoiding intermediate files has several advantages:

- Improves the simulation execution times, because data takes time to be written.
- No post-processing is necessary because the final result is directly available at the end of the simulation runs.
- The study is simpler from a user's point of view because, once the configuration files are ready, the only action to perform is calling the launcher script.

We believe that these reasons go further than a quantitative study. Melissa is in fact a game-changer that allows performing studies that were not possible before, and simplifies the realization of the existing studies.

MetaModel Training

We led a series of developments and experiments to propose a novel way to compute deep surrogates by online training rather than the classical file-based approach. This led to a new version of the Melissa server

Experimental results obtained go beyond our expectations with significant gains in both performance and learning quality compared to the classical offline training strategy where simulations are first executed to produce, and next train the neural network through several epochs reading the simulation data from disks. The table below is an extract from our publication at Supercomputing'23. The offline line operates in two phases 1) data generation with 2000 cores and storage in files 2) classic learning by epoch with 4 GPUs. Even when configuring Pytorch to overlap file reads, the number of samples processed by the GPU is 38 samples/s. The online line generates the data (5000 cores) and does the learning on 4 GPUs at the same time. Learning goes from 24 hours offline to less than 2 hours with a processing rate of almost 500 samples/s. With no more storage constraints, learning is done on 2M examples (10TB of data) instead of 25k offline (100GB), improving the generalization capacity of the neural network by more than 45%.

Buffer	Generation/Training Resources(Cores & Gpu)	Generation (hours)	Total (hours)	Dataset (GB)	Unique Samples (N)	RMSE ↓	Gain (%)	Throughput (samples/sec)
Offline	2,000C / 40C, 4G	0.22	24.5	100	25,000	25.1	—	38.2
Reservoir	5,120C / 40C, 4G	—	1.97	10,000	2,000,000	13.2	47.4	476.7

Using consolidated figures provided by the supercomputer center (1 kh/core CPU = 6€, 1 kh/GPU V100 = 360€, 1TB (SSD storage) = 56€), leads to a cost with online training at 63.8€, only 29% above the cost of offline data generation and training at 49.1€. The cost of offline training would decrease to 41.16€ when repeated (no storage and data generation costs). If offline training would have been performed with the 8TB dataset of online training, the sole storage cost would account for 480€.

This work meets REGALE Strategic Objectives SO1.1 Improved application performance with execution time going from 24h to 2h (x12), SO1.4. Melissa has been designed to address SO2 Scalability (no file thus no I/O bottleneck and better usage of GPUs), SO2 Platform Independence (Pytorch and Tensorflow are both supported) and SO2 Extensibility (modular design easing extensions to other simulation, support for various batch schedulers, data processing engines). SO3 objectives are also addresses: SO3 Automatic allocation of resources, Melissa taking care of interacting directly with the batch scheduler (SLURM and OAR supported) for resource allocation; SO3 Programmability switching from C original implementation of the server to Python for ease of use; S03 Flexibility thanks to Melissa modular design.

6.3 Pilot 3: Enterprise Risk Assessment

Scope: OLISTIC Enterprise Risk Management Platform provides risk assessment for organisations of various sizes. Risk assessment in OLISTIC is based on a model that represents assets and their vulnerabilities, and a graph (directed acyclic graph (DAG) that depicts all applicable asset interconnections. To calculate the overall risk assessment we need to identify and quantify the specific risks associated with each software component and their interconnections. Furthermore, we use asset chains that represent every possible path among software assets in order to calculate the attack paths that can be exploited to reach specific assets. This is a unique selling point and major benefit of OLISTIC that allows organizations to better protect even their most critical assets.

State-of-practice: Until now, OLISTIC operated with a functional version that included attack path calculation and risk assessment, through the usage of Drools business rule management; however the operation was limited by the size of the attack path and the number of asset vulnerabilities . Finding all available paths in a graph can have exponential time complexity, which makes calculations even more difficult as the topology increases. The same is true the analysis of low-level traffic collected from (net-traffic-)agents is required and can be very demanding when the magnitude of data scales, resulting in high-CPU usages and can be time-consuming for the retrieval of computed risks. With the transition to an approach based on the usage of optimised traversal algorithms and the usage of High-Performance Computing (HPC), we enhance the platform's efficiency significantly, streamlining the assessment process, in comparison to existing server setups that cannot easily support such magnitude of processing capabilities.

REGALE value proposition: Utilization of an HPC environment for the network analytics process for Advanced Persistent Threats and graph calculations, can efficiently lead to higher performance results. In the context of REGALE we extend the OLISTIC platform accordingly for retrieval of high-volume traffic in batch-processing manner, to allow the placement of network-metadata processing workflows to the HPC environment leading to significant performance gains.

More specifically, the following areas of interest are examined by utilizing the Ryax Platform:

- Efficient placement of reconfigurable programming modules according to input parameters indicating the retrieval periods of the data. Thus utilization of a framework supporting the interchangeability of processing workflows according to our needs.
- Enchantment of system capabilities by placing the workflow modules in applicable nodes. Making it possible to scale Spark analytics workflows with regard to the volume of traffic collected and thus increase the consumed/processed data over time.

Evaluation scenario: The main objective of this analysis is to calculate all potential asset vulnerabilities for each chain in a graph (Figure 6.1), where each chain represents every possible path among software assets. By systematically evaluating each path, we can identify and quantify the specific risks associated with each software component and their interconnections. This comprehensive approach allows for a more precise understanding of where security measures need to be strengthened within the software architecture. Such a detailed vulnerability assessment is critical for enhancing the overall security posture of the system, ensuring that protective efforts are both efficient and effective in mitigating potential threats.

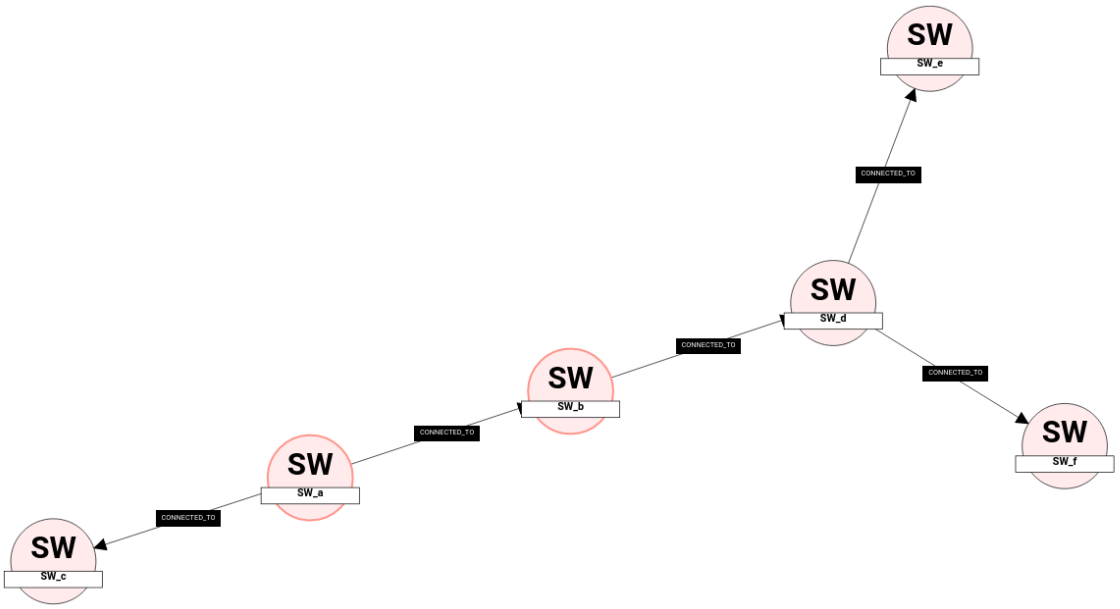


Figure 6.1: Graph of Software Assets and their Interconnections

In order to calculate the asset vulnerabilities for each chain in the graph, we must first identify each chain by employing graph traversal algorithms. Techniques such as Depth-First Search (DFS) or Breadth-First Search (BFS) can be utilized to explore and enumerate every possible path between the software assets. Following the identification of these chains, the next step

involves calculating the asset vulnerabilities. This is achieved by using the Cartesian product for each asset within the paths. By combining every possible pairing of assets, we can thoroughly analyze and understand the potential vulnerabilities that may arise from different interactions between software components. This methodical approach ensures a complete and nuanced assessment of risks throughout the network.

Consider two interconnected software assets, SWa and $SW\beta$, each with their own sets of vulnerabilities. Let's say SWa has vulnerabilities $\{CVE-0001, CVE-0002, ..., CVE-000n\}$and $SW\beta$ has vulnerabilities $\{CVE-0001, CVE-0002, ..., CVE-000m\}$.

The Cartesian product of the asset vulnerabilities of $SWa \times SW\beta$, denoted as $\{(CVE-0001, CVE-0002), (CVE-0001, CVE-000m), ... (CVE-000n, CVE-000m)\}$ involves pairing each vulnerability of SWa with each vulnerability of SWb. This results in a set of ordered pairs, where each pair consists of one vulnerability from SWa and one from $SW\beta$.

Experimentation and Validation: This section delves into the evaluation process of the proposed methodology for assessing software asset vulnerabilities using graph traversal techniques that have been implemented. We outline the experimentation setup, describe the computing infrastructure used for model deployment, and discuss the specific data utilized in our experiments. Additionally, we detail the implementation adjustments made for efficient data handling and processing.

Experimentation Setup

The experiments were conducted on the High-Performance Computing (HPC) infrastructure of the Grid5000 supercomputer. Grid5000 is a large-scale and versatile testbed, providing access to a substantial amount of resources distributed across multiple sites in France. This infrastructure is ideal for demanding data processing tasks and was leveraged to deploy and evaluate our model effectively.

Data Description

For our experiments, we utilized data stored in tables from an RDBMS, specifically designed for this study and referred to as the OLISTIC DB (explained in more detail in deliverable D4.3). The key tables used include:

Table 6.3: Experimentation Data Utilized

Table Name	Description	Number of Records
Asset	Asset information including type (software, hardware, peripherals, network), CP23 name, and Fully Qualified Domain Name (FQDN)	417
AssetAssetRelationship	Describes the interconnections between assets	326

AssetRelationship	Specifies the types of relationships between interconnected assets	379
Vulnerability	Details on vulnerabilities including name, access complexity, access vector (LOCAL/NETWORK), availability impact, CVSS version, CVSS score, brief description, exploitability score, and impact score	229017
VulnerabilityV31	Updated vulnerability details following CVSS v3.1 standards, including name, attack complexity, attack vector (LOCAL/NETWORK), availability impact, and other relevant info	112968

In Table 6.3, the data that was instrumental in forming the dataframe for constructing the graph of software assets can be described.

The constructed graph, derived from the OLISTIC database, features 13 distinct entry points / initial assets. These entry points serve as the starting locations for the graph traversal algorithm. From these initial positions, a total of 2,954 chains were discovered. Each chain comprises up to 16 assets, demonstrating the complexity and interconnected nature of the assets represented in the graph. This structure facilitates a comprehensive analysis of potential vulnerabilities within the network, leveraging the detailed interconnections mapped out through the data.

Implementation Details: In the execution phase, instead of performing relational queries directly within the Olistic DB, we employed Spark SQL. This approach allowed us to execute the necessary operations efficiently during the graph traversal process. By leveraging Spark SQL, we were able to handle large volumes of data seamlessly and expedite the computation of asset vulnerabilities throughout the software chains.

The experimental phase involved several key activities (see D4.3):

1. Graph Construction: We first constructed a graph representing the software assets based on data extracted from the OLISTIC DB.
2. Graph Traversal: Using graph traversal algorithm, we explored the graph to identify distinct chains of software assets, initiating the traversal from specified entry points or indices.
3. Vulnerability Simulation and Analysis: For each identified software chain, we simulated asset vulnerabilities by assuming that each asset could have a variable

number of vulnerabilities, ranging from 1 to n. This simulation used a pool of real vulnerabilities sourced from Common Vulnerabilities and Exposures (CVE) listings.

About CVEs

The Common Vulnerabilities and Exposures system catalogs publicly known security vulnerabilities. Each record in this catalog has a unique identifier and includes details such as a brief description and, when possible, remediation steps. This catalog is acknowledged across the security field, making it suitable for authentic simulations in security studies.

For our experiments, we query a random set of real vulnerabilities from the CVE database to assign an arbitrary number of vulnerabilities (from 1 to n) to each asset in the software chains. This approach ensures that our vulnerability assessment is grounded in real-world scenarios, enhancing the relevance and applicability of our findings.

An example for the described procedure is:
Let A be a set representing all assets in a software chain, such that: $A = \{a_1, a_2, a_3, \ldots, a_m\}$ where m is the total number of assets in the chain.

For each asset a_i in A, let V_i be the set of vulnerabilities associated with a_i. The number of vulnerabilities of each asset a_i can vary from 1 to n. Thus, we can define V_i as: $V_i = \{v_{i1}, v_{i2}, v_{i3}, \ldots, v_{ik}\}$ where k is the number of vulnerabilities for asset a_i and $1 \leq k \leq n$.

Given that you are interested in exploring every possible pairing of vulnerabilities across different assets to thoroughly analyze potential interactions and risks, the mathematical representation using the Cartesian product would be: $V_1 \times V_2 \times \ldots \times V_m$ where each element of the Cartesian product represents a tuple: $\left(v_{1j1}, v_{2j2}, \ldots, v_{mjm}\right)$ with v_{ij} being a vulnerability of asset a_i.

Results

This section outlines the outcomes of the scalability and application-specific experiments conducted to assess the performance of our methodology.
We conducted scalability tests using different configurations of executors to evaluate how the system performs under varying loads. These tests involved configurations with 1, 4, 8, 10, 20, 30, 40, 50, 60, 70, and 80 executors, each equipped with 8 GB of memory and 2 CPU cores. This diverse range of settings allowed us to systematically measure the impact of resource allocation on the performance of our graph traversal algorithms across extensive asset networks.
Two distinct experiments were performed to validate the application's effectiveness in different scenarios:

- Full Graph Analysis: In this experiment, the entire graph was analyzed starting from all entry points (traversing all 2,954 chains). Each asset in the graph was assigned three vulnerabilities. This comprehensive approach tested the model's capacity to handle large datasets and complex interactions simultaneously.

- Targeted Path Analysis: The second experiment focused on targeted path analysis, starting from three specific entry points. This approach utilized 22 chains for assessing exploitability, with each asset along these paths having five vulnerabilities. This experiment aimed to simulate a more focused attack scenario, testing the system's ability to identify and evaluate vulnerabilities in a constrained yet critical subset of the network.

In Table 6.4 we present the execution times for both the full graph analysis and the targeted path analysis experiments in a clear and concise manner.

Table 6.4: Execution Time

Executors	Full Graph Execution Time (s)	Targeted Path Execution Time (s)
1	1053.27	75.12
4	380.19	82.81
8	216.86	62.49
10	461.91	60.93
20	627.80	68.49
30	638.65	71.85
40	463.14	73.63
50	394.04	77.63
60	740.81	84.64
70	334.10	90.05
80	382.30	97.31

The execution times for both the full graph analysis and the targeted path analysis experiments are also depicted in Figure 6.2.

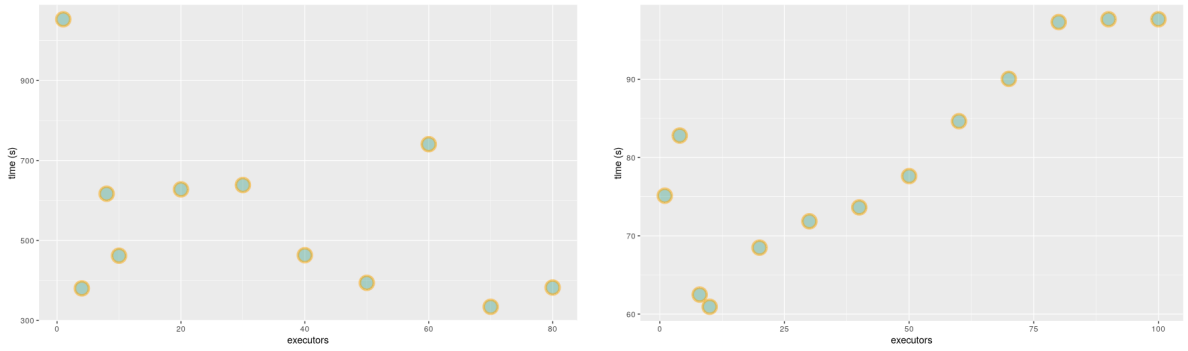


Figure 6.2: Visual Representation of Execution

6.4 Pilot 4: Complex geomorphometric models executed over Scope: High-precision, multi-factor models using earth observation data for groundwater estimation and very large data volumes management

State-of-practice: Geomorphometric models perform a number of operations on Digital Elevation Maps (DEMs) in order to calculate factors like hydrological flow directions and water pooling and produce the relevant maps. The construction of a workflow able to produce high-precision results requires the combination of different actions with significant computational load, that involve fluid dynamics and thus require Computational Fluid Dynamics (CFD) methods to be solved. The pre-REGALE implementation of the groundwater estimation and management service operates over relatively small land areas and uses the single-threaded implementation of the Open-geomorphometry toolset. While it produces sufficiently accurate results, its performance can be significantly improved by being able to handle larger land areas, and - on the usability side, being able to quickly run different configurations of the service by fine-tuning the execution parameters and customizing the classification settings for different areas.

REGALE value proposition: The aforementioned barriers to provide high-precision groundwater estimation services can be overcome via the solutions provided by REGALE, and exploited in the context of agricultural operations, environmental research and policy making. Specifically, in the context of REGALE, we aim to:

- Increase the capability of the system with respect to the analysis of digital elevation maps that can be feasibly used as input to the described models
- Increase coverage, that is the land area that can be covered in acceptable computation times
- Optimize workflow execution in terms of data processing, intermediate result production and transferring and inter-process communication in the context of the service.

Evaluation scenario: The assessment of the benefits obtained via the usage of REGALE solutions will focus on two aspects affecting the quality of the system: response times for a single workflow, and ability to run multiple workflows with different configurations. On the first part, the core metric to be used is the time to completion of the workflow. The baseline will be the time to completion achieved in the execution on a local small-scale server (32 GB of memory, 16 cores), for maps of three different sizes. For the second part, connected to the usability of the system, the metric that will be used is the time of configuration, deployment, and execution of at least 3 different configurations of the service for a given map. The selection of the map for the second evaluation branch will depend on the results observed for the first part. A map where the execution time benefits were relatively smaller will be selected, to better estimate the gains from using RYAX for managing and configuring the execution of the different workflow configurations.

Experimental Results: Our experiments were conducted on a DEM from Epirus, an area in north-west Greece.

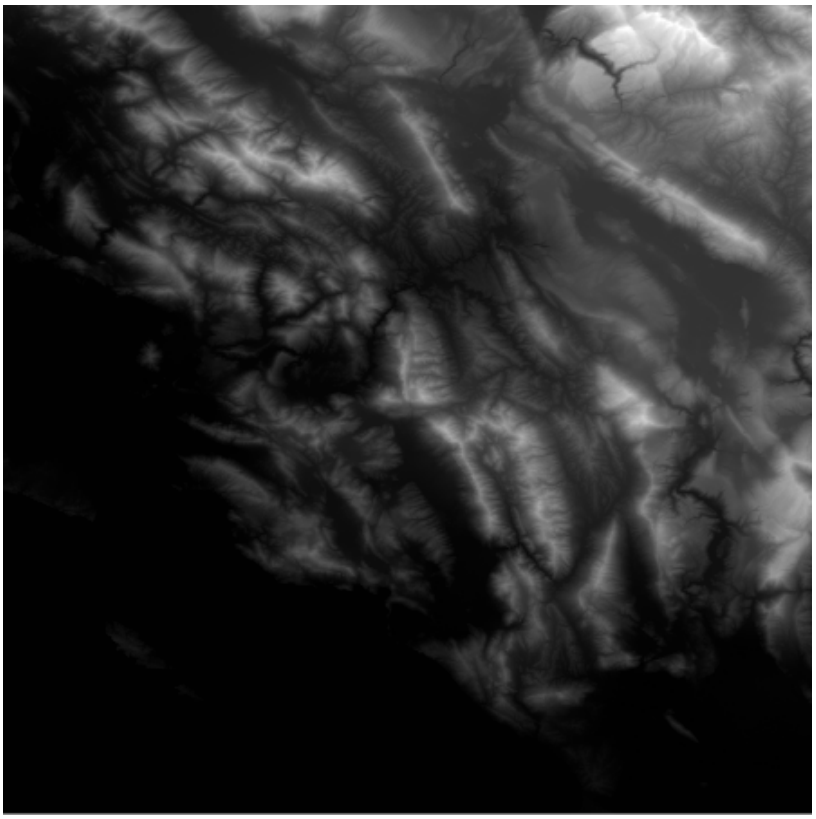


Figure 6.3: Initial DEM file from Northwest Greece

The original DEM (Figure 6.3) had a resolution of 3600x3600 pixels (~13 million pixels), covering more than 7.000 km^2. In order to evaluate how the increase in data points affects the total execution time of the algorithm and how the use of parallelizing techniques speed up the process, experiments with various sub-areas were conducted with incremental size. Table 6.5 shows 5 cases that were processed along with their calculation time for each of the three components of the Geomorphometry tool.

Table 6.5: Cases considered

Sub area	Resolution	# of pixels	FlowMapR (s)	FormMapR (s)	FacetMapR (s) (NO spark)	FacetMapR (s) (Spark)
Epirus 1	72x110	7920	234.5	560.3	217.8	36.8
Epirus 2	179x186	33294	1915.5	1661.2	918	153.8
Epirus 3	159x337	53583	3865.9	2624.3	1415.5	219.3
Epirus 4	265x339	89835	4960.1	3512.4	2392.3	354.1
Epirus 5	450x450	202500	166224.68	4417.96	11324.53	731.48

All five of the extracted DEMs were processed in the pipeline by FlowMapR and FormMapR to collect the necessary features for the last part. FacetMapR processed the outputs of the previous components in two distinct settings, namely with or without Spark. By enabling Spark, the algorithm is able to harvest the power of all the present CPU cores and distribute independent parts of the workload to other computation nodes of the network to further increase the processing power. It must be noted that the execution time is not only dependent on the size of the DEM but also on its unique landscape and characteristics. A

more complex landscape of a smaller resolution input file might exhibit processing time similar to a higher resolution input file that represents a simpler landscape. Figures 6.4-6.8 show the four cropped areas while Figure 6.9 shows all the DEMs overlayed in one image.

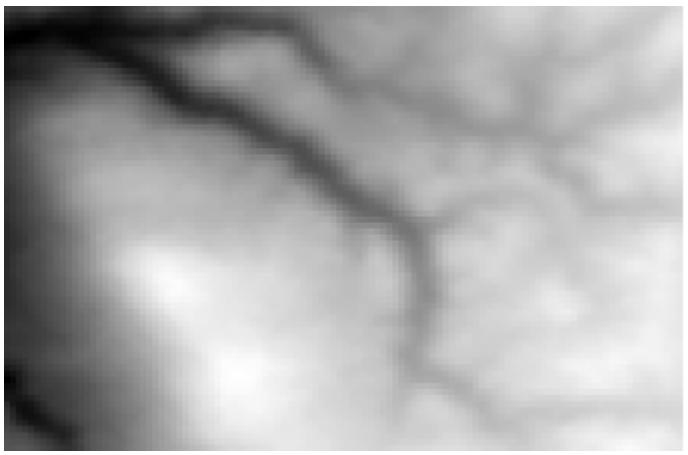


Figure 6.4: Epirus 1 area (72x110)

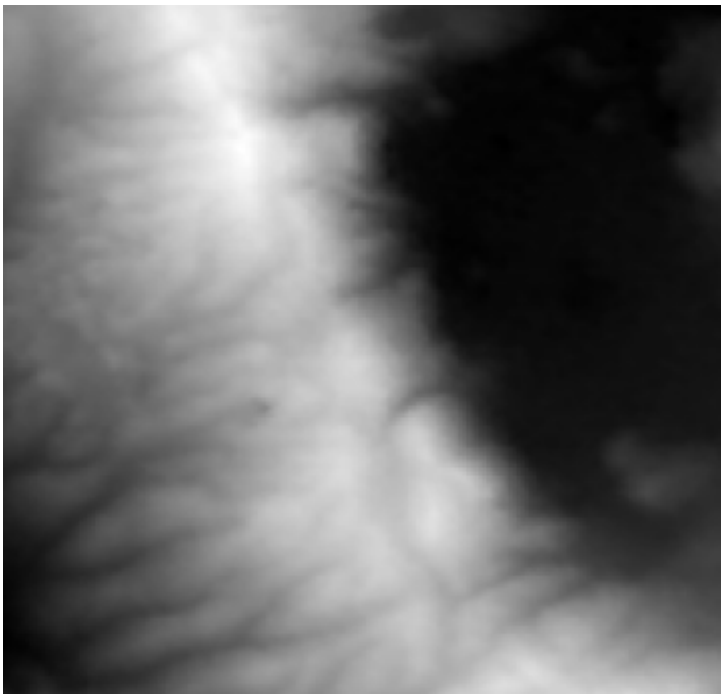


Figure 6.5: Epirus 2 area (179x186)

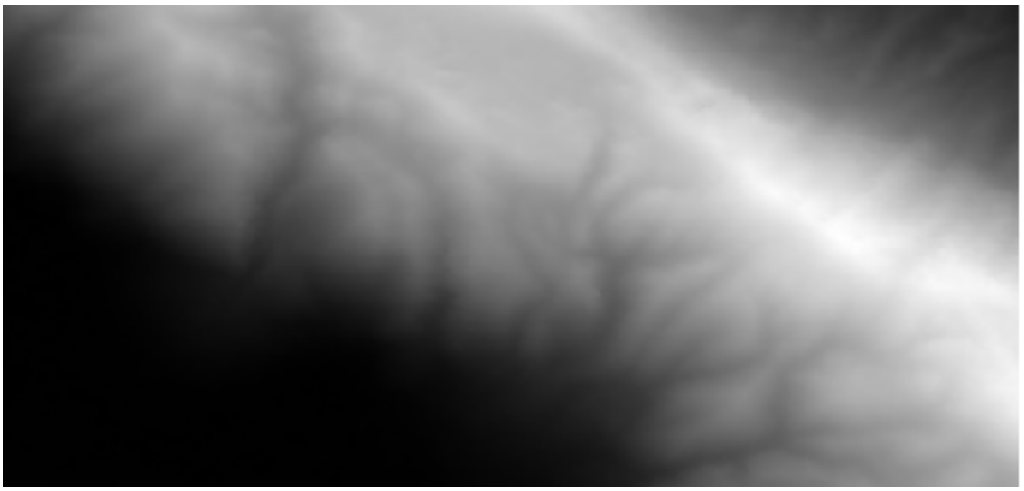


Figure 6.6: Epirus 3 area (159x337)

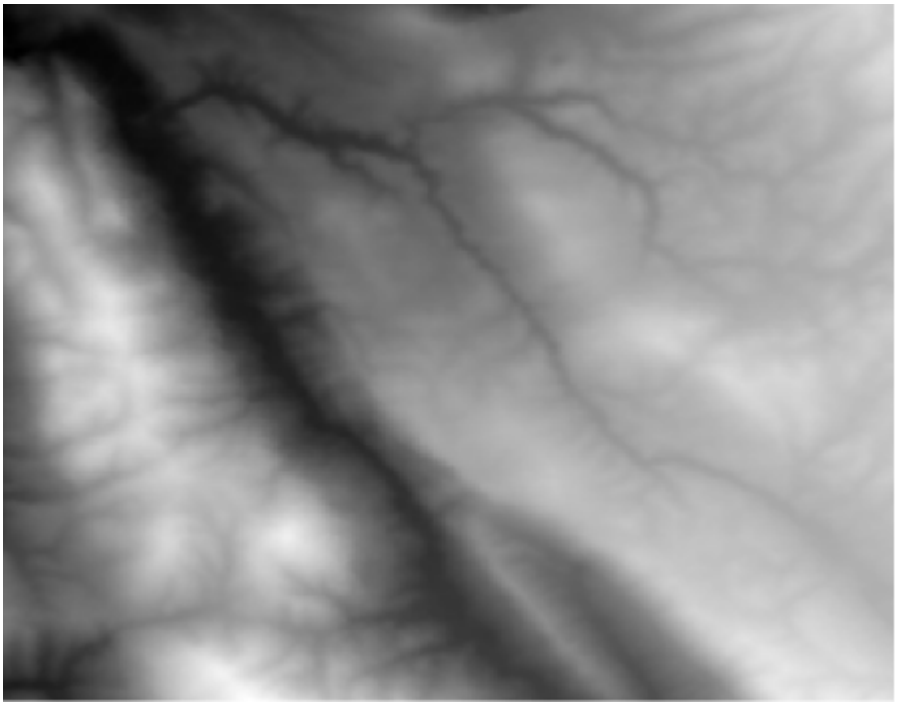


Figure 6.7: Epirus 4 area (265x339)

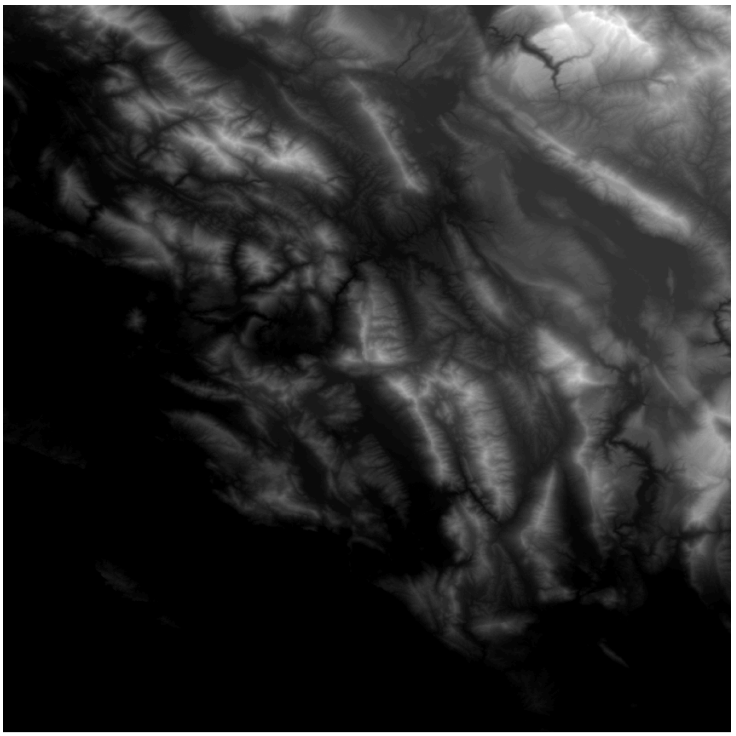


Figure 6.8: Epirus 5 area (450x450)

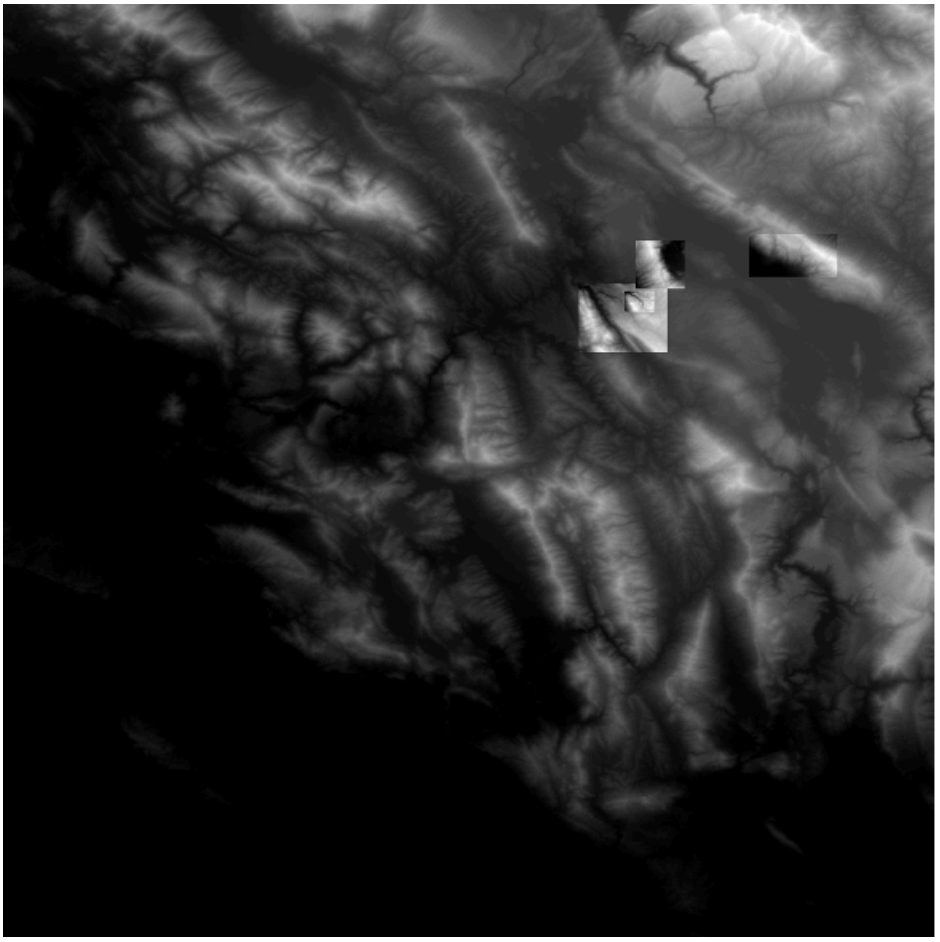


Figure 6.9: All DEM areas overlayed for easier comparison between them.

Furthermore, we examined different parallelisation configurations for the biggest use case (Epirus 5), to assess the scalability of the solution as available hardware resources increase. FacetMapR was executed over 1, 4, 10, 20, 30, 40, 50, 60, 70, 80, 90, 100, and 110 execution nodes, with the obtained results in execution time and speed up summarised in Figures 6.10 and 6.11.

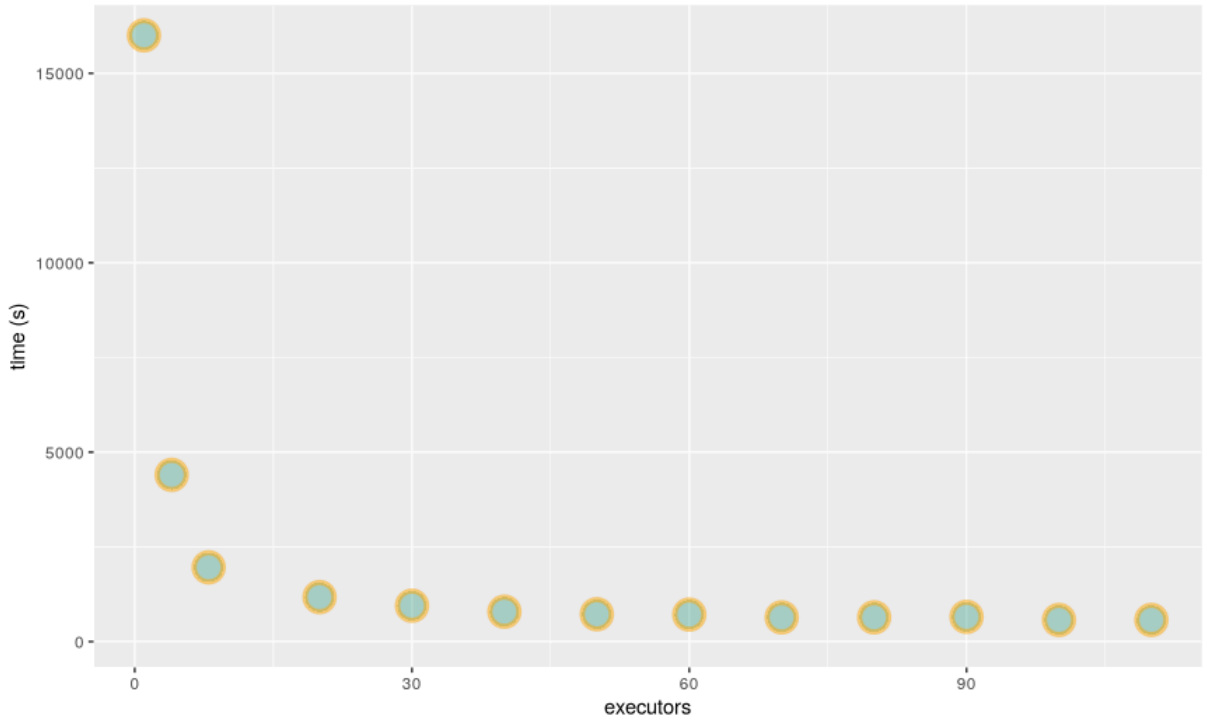


Figure 6.10: FacetMapR execution time relative to the number of available execution nodes

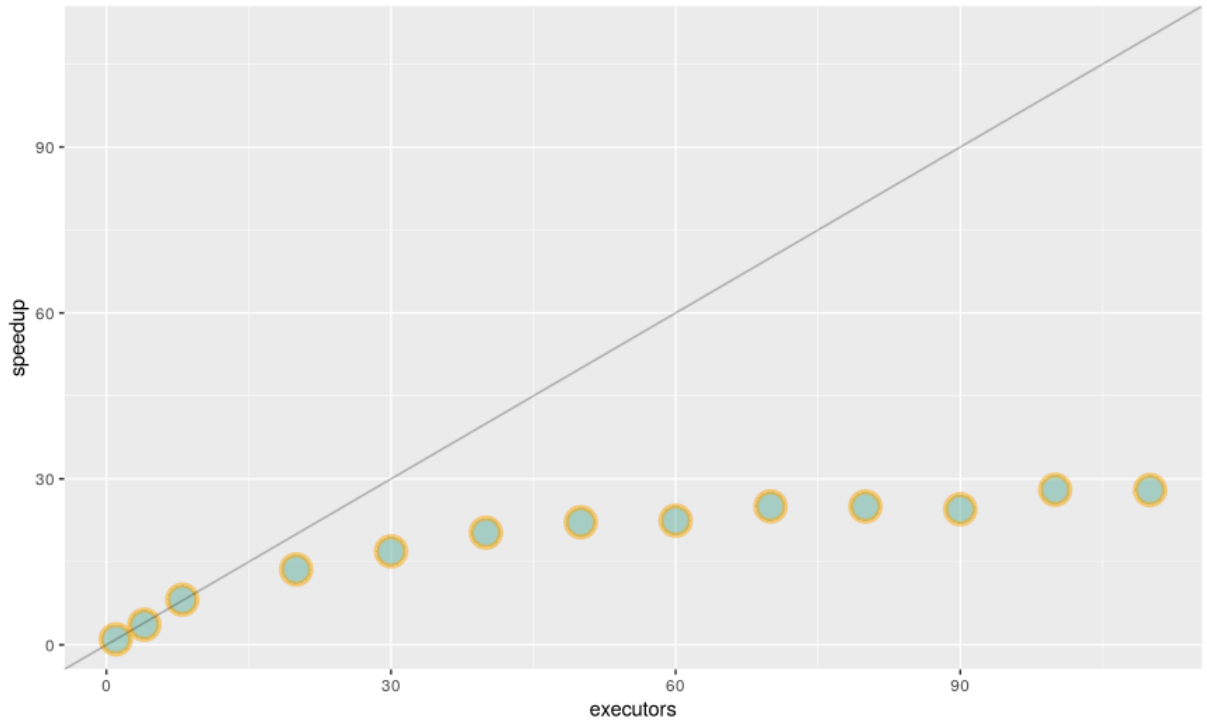


Figure 6.11: FacetMapR speedup relative to the number of available execution nodes

Another important, non-quantitative aspect pertaining to the overall performance gains via the adoption of REGALE solutions for the tool, is the ability to simultaneously run different

workflow configurations for the same map (Figure 6.12). Through the RYAX platform, multiple executions on the same input data (flow and form outputs) but with different hyperparameters (crule, arule etc.) can be run in parallel instead of sequentially, thus eliminating the need of waiting for their serial execution and therefore needing essentially the sum of the different execution times to properly assess the best solution for a given estimation problem.

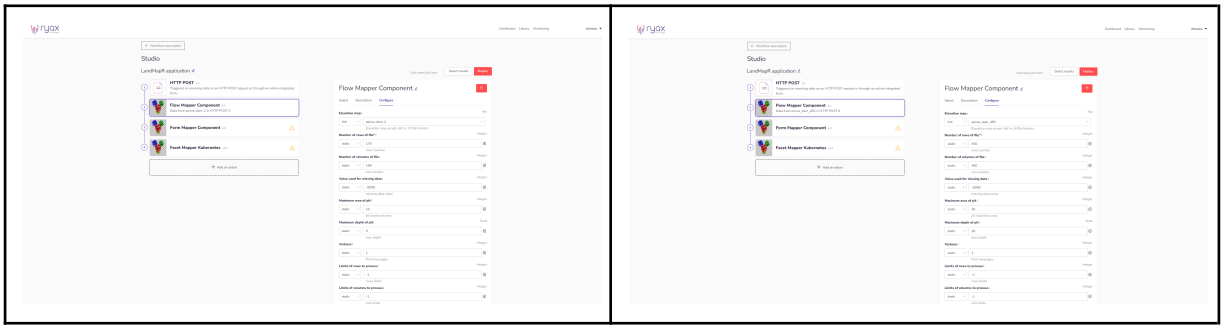


Figure 6.12: Workflow definition with different configurations for concurrent execution

6.5 Pilot 5: Design of car bumper made of carbon nanotube reinforced polymers

Scope: Improve performance of stochastic optimization algorithms in complex multiscale models.

State-of-the-practice: The goal of this pilot is to design an innovative car bumper made of carbon nanotube (CNT) reinforced polymers. To achieve this, an optimization problem needs to be solved, where the goal is to find the optimal weight fraction of CNTs and/or their orientation (design variables for the problem) within the polymeric matrix that will lead to enhanced crashworthiness of the part. In addition, the problem is formulated in a stochastic setting, where the randomness in the material properties and the loading conditions is taken into account for a more rational design. The solution to this stochastic optimization problem requires the generation of a large number of instances for the design variables and for each one, a separate Monte Carlo simulation needs to be performed in order to evaluate the statistics of the response of the bumper in crash scenarios.

The aforementioned solution framework for this pilot application has been conceptualized but its implementation has never been attempted before the REGALE project. The reasons for this are:

- The immense computational requirements for performing massive numbers of model simulations, with each simulation taking several hours to complete.
- The data storage and processing requirements for computing the statistics of the car bumper's response at each time instance of the crash simulation.

REGALE value proposition: The REGALE project offers solutions at multiple levels in order to overcome the computational barriers associated with this pilot application. Specifically, the REGALE tools facilitates the integration of various modules (e.g. monitors, node managers, job managers) with our in-house code for an efficient deployment on supercomputers and ensures the optimal resource allocation for our application, while maximizing throughput. In this direction, the Melissa workflow manager provides a means of performing 'on-the-fly' computation of statistics for the problem's response, thus drastically reducing the storage requirements and the cost for data processing.

Evaluation scenarios: To assess the benefits offered by the REGALE project, we focused on two scenarios.

- *Scenario 1*: *Execution on a single node vs Execution on a supercomputer*
  - Baseline: The time to completion achieved in the execution on a single core (from a node consisting of 1 CPU Intel Xeon Gold 5220, 18 cores/CPU, 96GB RAM).
  - KPI: Application speedup. The system throughput (jobs/hr) for the execution on a single core and the execution on a supercomputer using multiple nodes will be compared.

- *Scenario 2*: *Execution on a supercomputer with Melissa vs without Melissa workflow manager*.
  - Baseline: The time to completion and the storage requirements for the pilot without the integration of the Melissa workflow manager.
  - KPI 1: Application speedup. The time to completion with and without Melissa will be compared.
  - KPI 2: Storage requirements. The reduction in the storage requirements through the use of Melissa will be evaluated.
  - Developer experience on Programmability: The ease of coupling the Melissa workflow manager with our in-house software will be assessed in a qualitative manner.

Experimental Results: The experiments involved two computational models, (i) a simple bumper model (Figure 6.13) that consists of 15402 displacement unknowns (system of 15402 linear equations), which was solved for 50 time increments and (ii) a more sophisticated bumper model (Figure 6.14) that consists of 658446 displacement unknowns (system of 658446 linear equations), which was solved for 20 time increments.

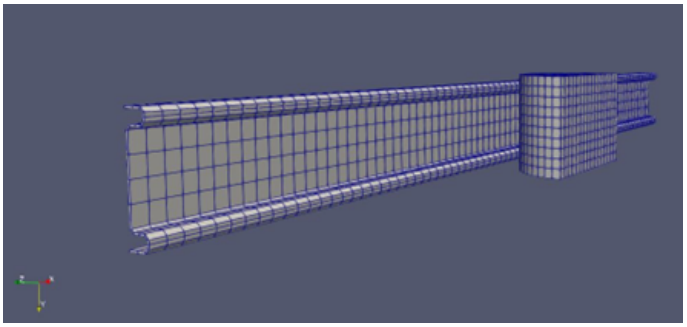


Figure 6.13: Finite element mesh of the first bumper model (15402 unknown variables)

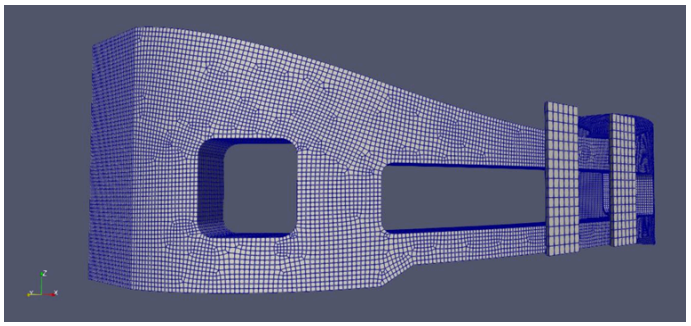


Figure 6.14: Finite element mesh of the second bumper model (658446 unknown variables)

For both bumper models, the stochastic optimization analyses required 50 optimization steps, where at each of these steps a population size of 10 was considered. To evaluate the fitness function at each optimization step, an additional Monte Carlo (MC) simulation was performed using 100 samples. Therefore, the total number of model runs required to derive the optimal material microstructures that would lead to enhanced crashworthiness was

$$50\ (No.\ of\ optimization\ steps)\ \times\ 10\ (population\ size)\ \times\ 100\ (MC\ samples)\ =\ 50000\ model\ runs$$

However, each model run required 3103,7 sec on a single core of a node for the first bumper model and 38520,8 sec for the second bumper model. It becomes evident that performing these types of analyses on a single machine would be computationally prohibitive. For this reason, we deployed our code on the Grid5000 supercomputer (https://www.grid5000.fr/w/Grid5000:Home) to accelerate the solution process through multi-node scheduling. Specifically, we used the "Gros" cluster, which consists of 124 nodes (1 CPU Intel Xeon Gold 5220, 18 cores/CPU, 96GB RAM, 447GB SSD, 894GB SSD, 2 x 25Gb Ethernet). In addition, to further reduce the cost, we utilized the Melissa workflow manager, which reduces the cost of the statistical post-processing in each of the MC simulations, as well as the data storage requirements.

The performance evaluation results are presented in Table 6.6 for the first bumper model and in Table 6.7 for the second.

Table 6.6: Performance evaluation for the first bumper model

	Computational cost (wall-clock time)				
	Single core	Distributed -without Melissa (100 nodes with 18 cores)	Speedup (wrt single core)	Distributed -with Melissa (100 nodes with 18 cores)	Speedup (wrt distributed architecture without Melissa)
One model run	3103.7 sec	-	-	-	-
One MC simulation (100 model runs + statistical post-processing)	311983 sec	3261.5 sec	x95.66	2912.1 sec	x1.12
One stochastic optimization step (10 concurrent MC simulations)	3119830 sec (estimate)	3472.8 sec	x898.36	3073.3 sec	x1.13
Total Analysis time (50 optimization steps)	43331 hrs (estimate)	48.3 hrs	x897.12	43.1 hrs	x1.12

Table 6.7: Performance evaluation for the second bumper model

	Computational cost (wall-clock time)				
	Single core	Distributed -without Melissa (100 nodes with 18 cores)	Speedup (wrt single core)	Distributed -with Melissa (100 node with 18 cores)	Speedup (wrt distributed architecture without Melissa)
One model run	38520.8 sec	-	-	-	-
One MC simulation (100 model runs + statistical post-processing)	1284 hrs (estimate)	43521.3 sec	x106.21	20724.3 sec	x2.10
One stochastic optimization step (10 concurrent MC simulations)	12840 hrs (estimate)	45012.6 sec	x1027	21935.8 sec	x2.05
Total Analysis time (50 optimization steps)	642000 hrs (estimate)	625.18 hrs	x1027	306.5 hrs	x2.04

As evidenced from the tables above, this work meets REGALE Strategic Objective *SO1.1 Improved application performance* with the speedup achieved for the first application being x897.12 and x1027 for the second. These numbers were close to theoretical values predicted by Gustafson's law since every optimization step is embarrassingly parallelizable. It is also interesting to notice that an additional speedup of x2.04 can be achieved for the second test case when using the Melissa workflow manager. This case has significant I/O read-write costs, while the statistical post-processing of the results during the MC simulation was also expensive. In this regard, Melissa proved very helpful at reducing these costs. On the other hand, the speedup offered by Melissa for the first test case was only x1.12, which is a reasonable value since the I/O communication and statistical post-processing costs are very small compared to the model evaluation cost for this less complex case. In terms of the storage requirements for the two applications, it is worth mentioning that a significant reduction was achieved with Melissa. In particular, for the second application, which was the most intensive, without Melissa each optimization step needed 106 GB for storing the data that would be later processed to extract the relevant statistics. However, with the use of Melissa the storage requirements were reduced to merely 0.2GB as the statistics were computed 'on-the-fly' and there was no need to store most of the analysis results.

7. REGALE sophistication

7.1 Multi-node co-scheduling

The state-of-practice in HPC resource allocation assumes entire CPUs (or NUMA nodes) to jobs. This leads to scalability issues for a large class of HPC applications that are memory constrained. Co-scheduling comes as a remedy to this problem by spreading memory-bound applications to more nodes (in the current setup doubling the size of nodes) and pairing (co-locating) them with other applications that are not memory bound and thus can synergistically co-exist with memory bound applications on the same CPU. Moreover, spreading HPC jobs in a large scale system to more nodes also changes the communication behavior, which may also have an impact on the overall performance. This ends up with jobs sharing nodes of the supercomputing system.

In our first set of experiments, we aim to study the potential of co-scheduling and measure the impact (speedup or slowdown) of applications being co-located on the same nodes for a specific amount of time. Based on these results, in our second set of experiments we simulate the co-scheduling of various application workloads and compare the results with the default “compact” policy. In our third set of experiments we test our adaptation of the OAR scheduler to support a version of co-scheduling in a small-scale system.

Results of application co-location: the potential of co-scheduling

We executed co-location experiments on the three machines described in Table 7.1. We utilized the two most popular benchmarks for parallel computing, i.e. the NPB[5] and SPEC[6] benchmark suites. We formulated several co-locations in each two benchmarks as follows:

- Benchmarks A and B are placed sharing the same node (the first half of all the cores of each CPU are assigned to benchmark A and the second half to benchmark B)
- The co-location lasts for 10 minutes during which if a benchmark finishes, it is restarted.
- We mix benchmarks of different process counts. This means that if a benchmark of x processes is co-located with a benchmark of x/n processes, the second benchmark is placed n times side-by-side with the first benchmark.
- In all cases we make sure that the initialization time (for job submission and MPI initialization) is kept low compared to the total experiment time.
- For each simulation we report the speedups of the involved jobs compared to the baseline (compact) placement

Table 7.1: Machines used for co-location

Supercomputer	**ARIS**	**Marconi**	**Grid5000 (DAHU/GRVINGT)**
Processor Type	Intel Xeon E5-2680v2 (Ivy Bridge), 2.8 GHz	Intel Xeon 8160 (SkyLake), 2.10 GHz	Intel Xeon Gold 6130, 2.10 GHz
Processors per Node	2	2	2
Cores per	10	24	16

[5] https://www.nas.nasa.gov/software/npb.html
[6] https://www.spec.org/hpc2021/

Processor			
Cores per Node	20	48	32
Hyperthreading	OFF	OFF	OFF
Memory per Node	64GB	196 GB	192 GiB
Network	Infiniband FDR, 56 Gb/s	Intel Omni-Path, 100 Gb/s	Intel Omni-Path, 100 Gb/s

Figures 7.1 - 7.5 show heatmaps of speedups of each benchmark when co-located with each one of the other benchmarks in the suite. In particular, Figure 7.1 focuses on a wide mix of small to medium sized jobs also mixed in different process count configurations. The fact that the majority of the collocations led to job speedups with an average of 1.13 provides the first strong evidence that co-scheding can be beneficial. Figure 7.2 provides the same information for the SPEC benchmarks suite demonstrating that the benefit of co-location was not an artifact of the NAS benchmarks suite but seems quite frequent among HPC applications. Figure 7.3 shows the results of larger jobs of 2048 MPI processes on the ARIS system which actually span almost the entire machine. Given the same positive picture, this experiment showcases that at least for this machine, jobs do not seem to slow down when they are spread in the interconnection network on this machine.

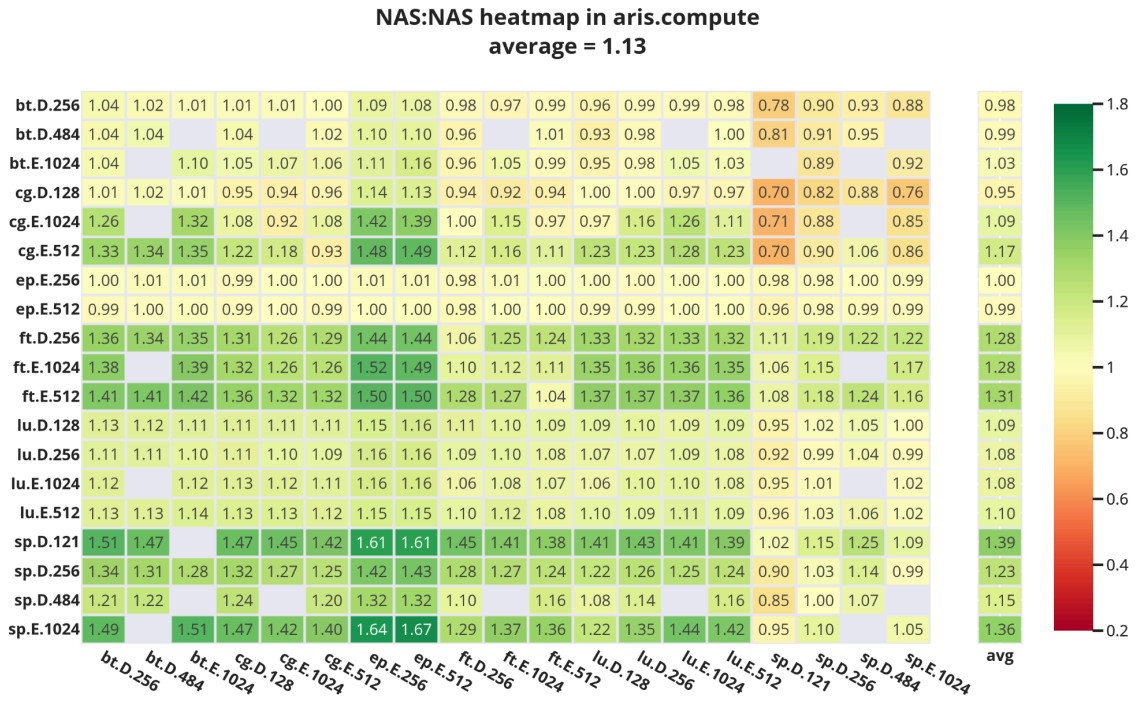


Figure 7.1: Co-location results (speedups) for the ARIS machine, NPB

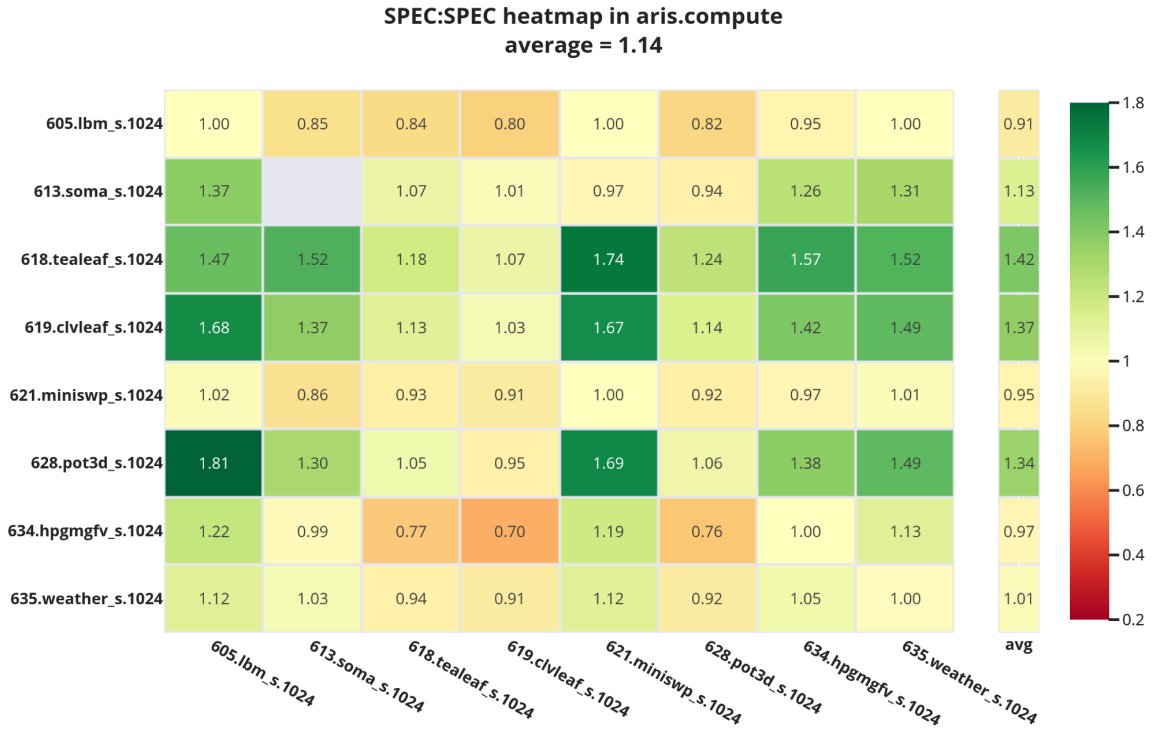


Figure 7.2: Co-location results (speedups) for the ARIS machine, SPEC

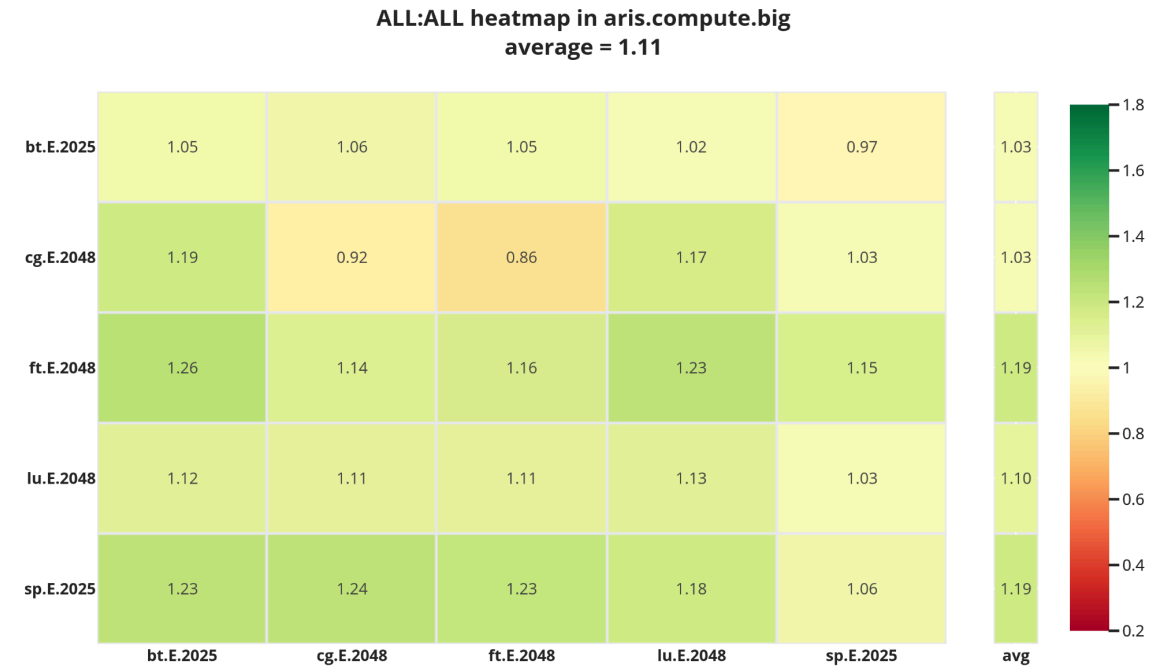


Figure 7.3: Co-location results (speedups) for the ARIS machine, NPB, large jobs

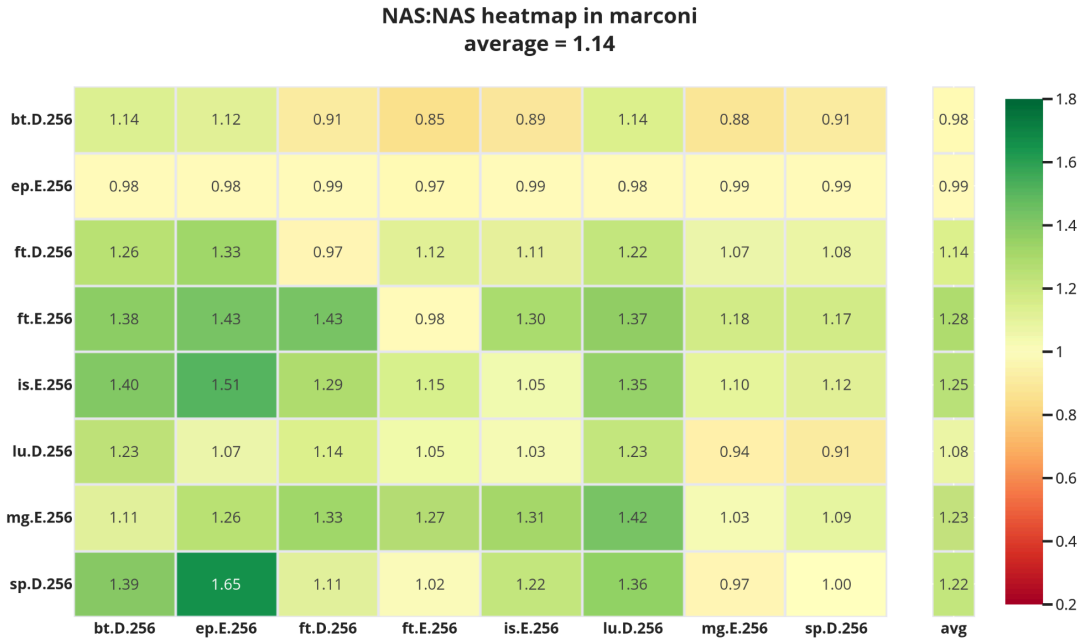


Figure 7.4 Co-location results (speedups) for the MARCONI machine, NPB

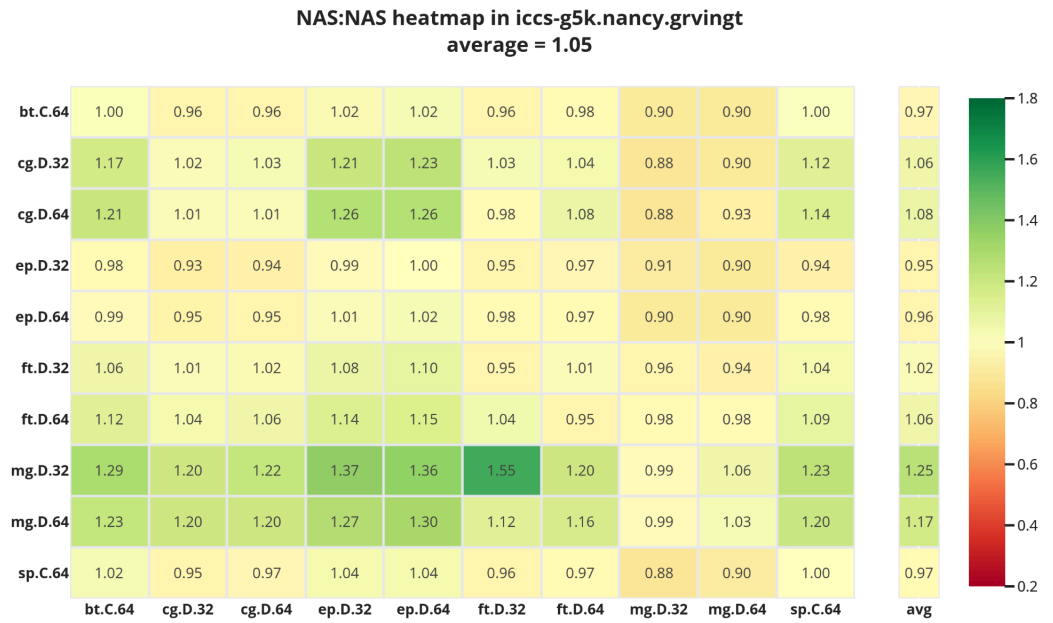


Figure 7.5: Co-location results (speedups) for the Grid5000 machine

Figures 7.4 and 7.5 demonstrate results from the other two machines. As for Marconi, a next generation machine compared to ARIS, the results are similar, demonstrating that the evolution of technology (larger core counts, faster interconnects) did not impact the behavior

of co-location and the potential of co-scheduling. Finally, the results with small-scale jobs on the Nancy machine of the Grid5000 infrastructure for small jobs show lower potential for co-scheduling, nevertheless again a positive prospect. Since we are granted root privileges in the Nancy machine, we will be using it to validate the co-scheduling adaptation we integrated in OAR (and described in Deliverable D2.3).

Results of co-scheduling simulation

In this set of experiments we utilize the heatmaps of the previous experiments and simulate the behavior of a co-scheduling heuristic algorithm for a large number of jobs. The simulation infrastructure was created for the needs of the REGALE project and is described in Appendix A. In each scenario we select a number of jobs from the NPB and SPEC benchmarks and execute multiple experiments on different machines. The heuristic algorithm assumes the existence of the heatmap presented in the previous paragraph. An approach to predict this heatmap is presented in Deliverable D2.3. For each simulation we report the speedups of the involved jobs compared to the baseline (compact) placement and makespan (total time to execution of the entire job set).

Figures 7.6 and 7.7 show the results of simulation for the two machines ARIS and Marconi respectively. Similar results were obtained with alternative configurations in the number of experiments and jobs. We may observe that even the random co-scheduling approach can provide a non-negligible makespan (throughput) improvement of 6-8% and with the more sophisticated co-scheduling heuristic we can reach a throughput improvement of 15% in ARIS and 42% in Marconi. This validates the accomplishment of the project objective of more than 15% throughput improvement as shown in Table 3.1.

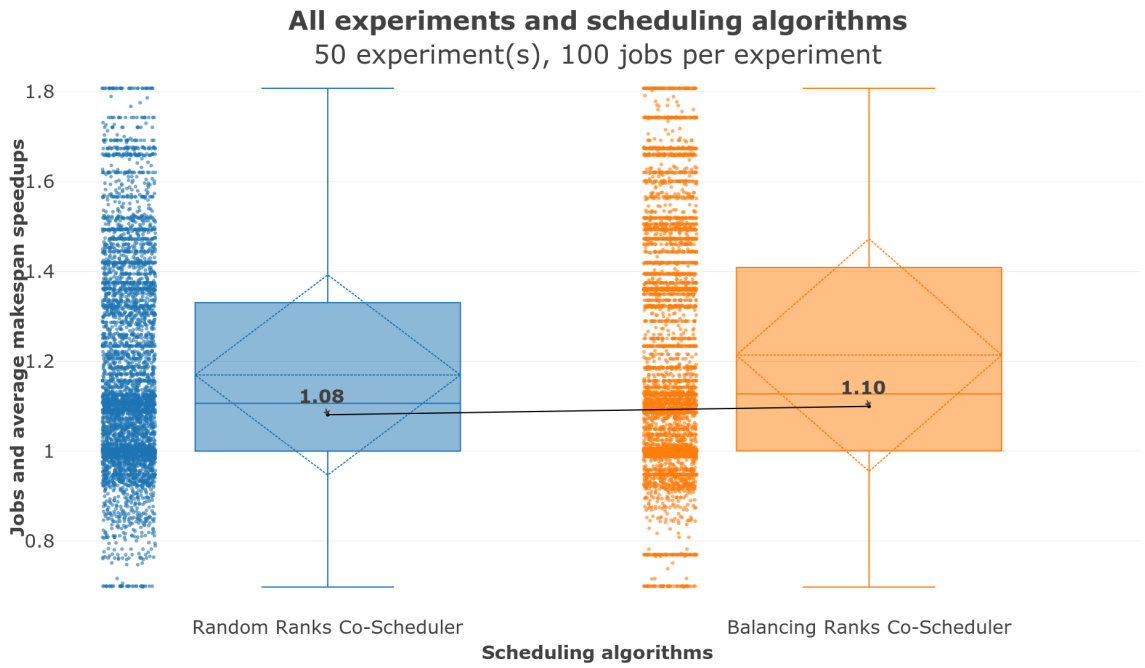


Figure 7.6: Simulation results for co-scheduling (Machine: ARIS, Number of experiments: 50, Number of jobs per experiment: 100, Co-scheduling algorithms: Random, Heuristics, Collected data-points: 10000)

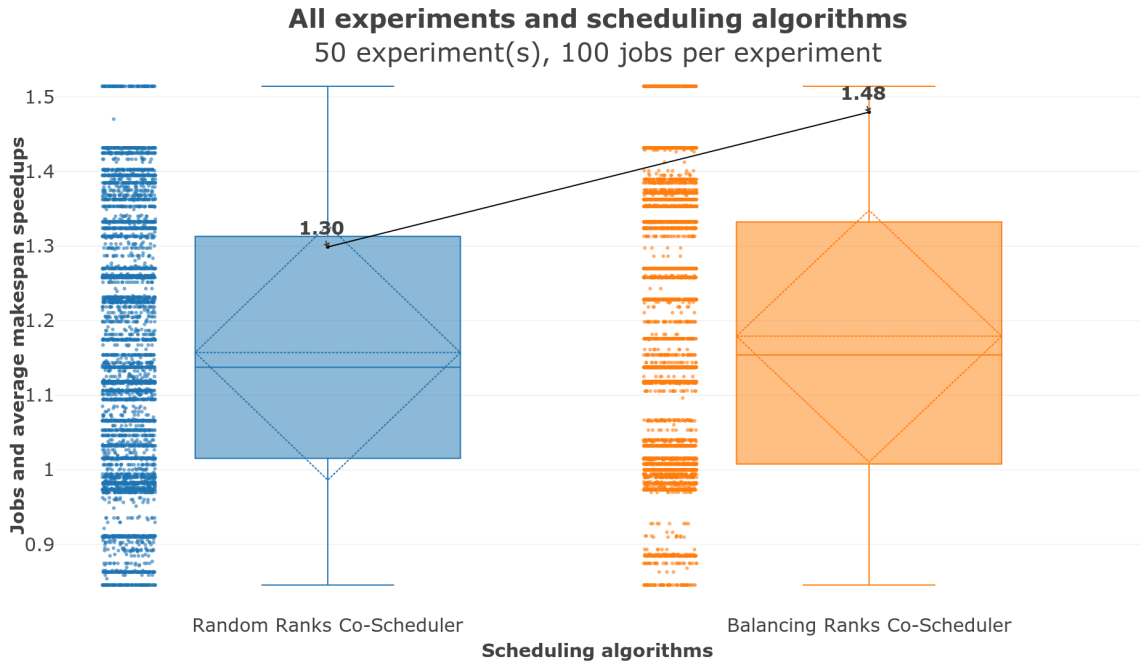


Figure 7.7: Simulation results for co-scheduling (Machine: Marconi, Number of experiments: 50, Number of jobs per experiment: 100, Co-scheduling algorithms: Random, Heuristics, Collected data-points: 10000)

Results with OAR

In this final set of experiments we present the results of compact scheduling, random co-scheduling and “heatmap-aware” co-scheduling as implemented in OAR and described in Deliverable D2.3. In the latter case the applications are tagged by the user based on their behavior presented in Figure 7.5 and the results are shown in Figures 7.8 and 7.9. We may notice again that even random co-scheduling is able to reduce the makespan of the execution and the heatmap-based selective policy is able to provide additional benefits.

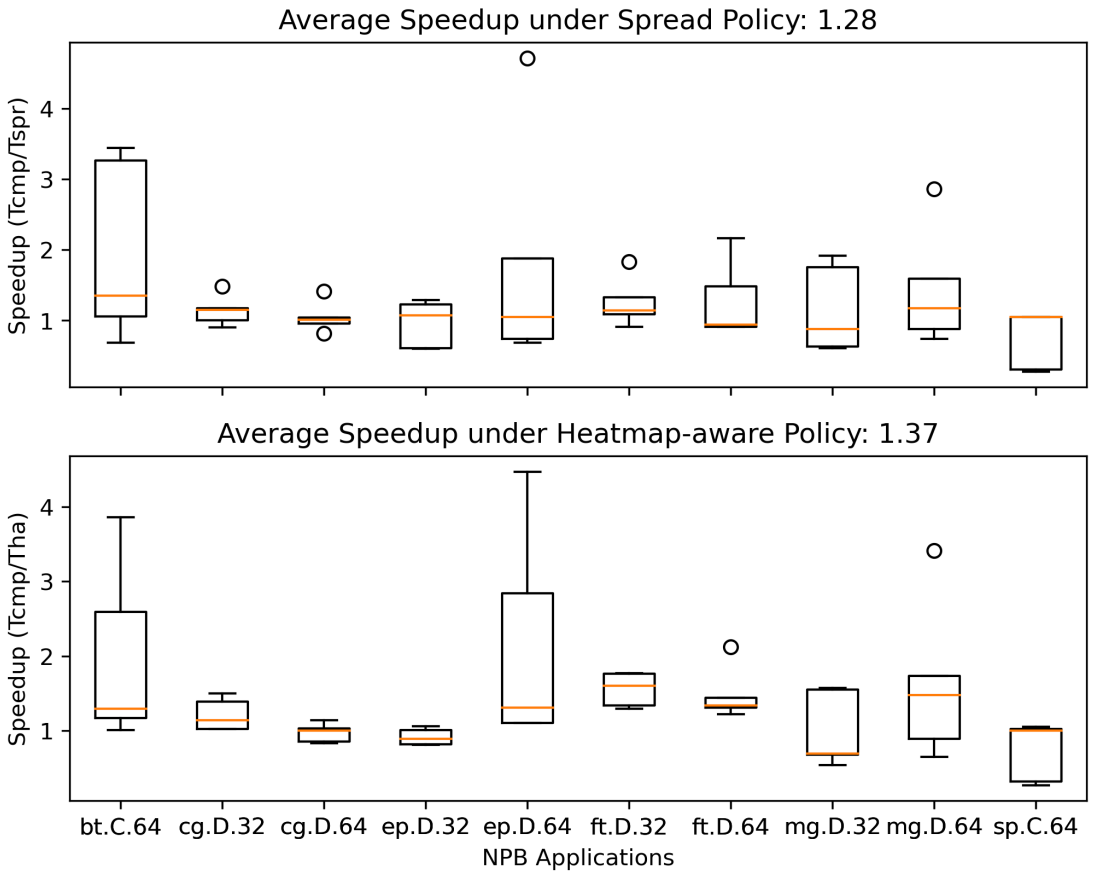


Figure 7.8: Speedup over compact for the spread and heatmap-aware policies with OAR. compact time = median of 5 compact runtimes

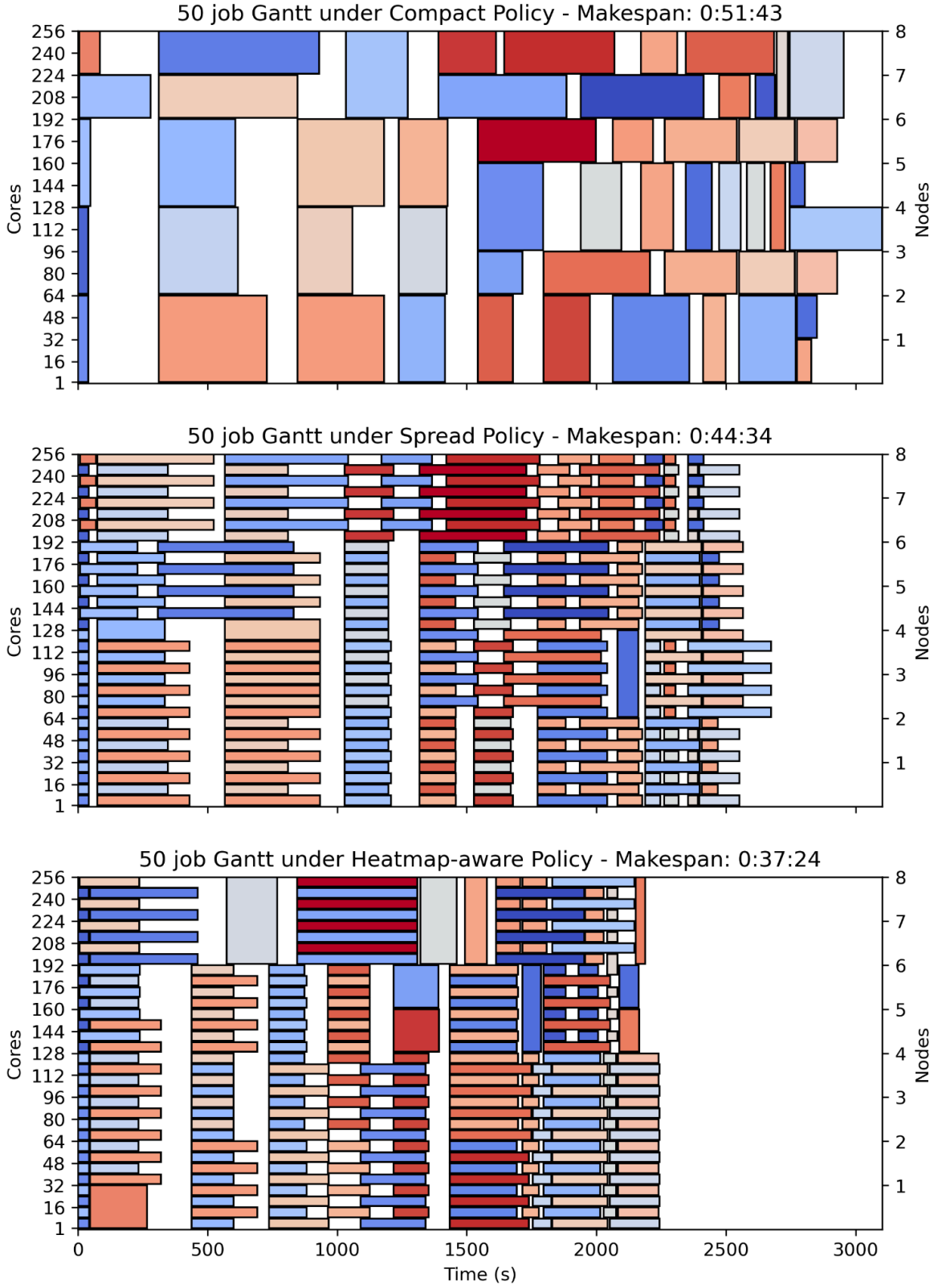


Figure 7.9: Compact, random co-scheduling and "heatmap-aware" co-scheduling with OAR. A dataset of 50 jobs from Grid5000-Grvingt Heatmap with equal selection (5 times) randomly distributed across the dataset, run under Grid5000-Dahu.

7.2 GPU co-scheduling

Scope: We target here job co-scheduling on compute nodes with GPUs, where each node GPU can be set to run different jobs. The expected benefit is a better usage of GPU resources, in terms of both device memory bandwidth and floating-point units. We test different co-location strategies, the most advanced one being based on reinforcement learning. For sharing a GPU between jobs we utilize the MPS and MIG features available for modern NVIDIA GPUs. The major objective of this sophistication is to improve throughput by 30% (SO2).

Workloads: We utilize the Rodinia benchmark suite, Stream/Random access benchmarks, and the Quicksilver mini application from CORAL benchmark suite. These well represent modern GPU workloads in the scientific computing area. We classify these benchmark programs into CI (Compute Intensive), MI (Memory Intensive), and US (UnScalable) as shown in the following table:

Class	Benchmarks
CI	lavaMD, huffman*, hotspot3D, hotspot*, heartwall*, bt_solver_A, bt_solver_B, bt_solver_C
MI	lud_A, lud_B, lud_C*, sp_solver_A, sp_solver_B, sp_solver_C, randomaccess, cfd*, gaussian*, stream
US	kmeans, dwt2d, needle*, pathfinder, backprop*, qs_Coral_P1, qs_Coral_P2, qs_NoFission*, qs_NoCollisions

We test our approach using different types of job mixes composed of the above programs: (1) CI-dominant; (2) MI-dominant; (3) US-dominant; and (4) Balanced. The exact job mix selections are as follows. Note the programs marked with * are unseen in the training phase of our approach.

Category	Name	Jobs
CI-dominant (CIx6, MIx3, USx3)	Q1	huffman*, bt_solver_C, bt_solver_B, hotspot3D, heartwall*, lavaMD, lud_B, cfd*, sp_solver_B, pathfinder, needle*, qs_NoFission*
	Q2	bt_solver_C, heartwall*, lavaMD, huffman*, hotspot*, hotspot3D, cfd*, sp_solver_C, gaussian*, pathfinder, needle*, qs_Coral_P1
	Q3	huffman*, bt_solver_C, hotspot3D, hotspot*, heartwall*, lavaMD, lud_B, stream, sp_solver_C, qs_NoFission*, pathfinder, needle*
MI-dominant (CIx3, MIx6, USx3)	Q4	bt_solver_B, heartwall*, bt_solver_C, lud_B, gaussian*, sp_solver_B, cfd*, sp_solver_C, stream, qs_NoCollisions, pathfinder, qs_Coral_P2
	Q5	heartwall*, hotspot*, bt_solver_B, lud_B, gaussian*, randomaccess, stream, lud_C*, sp_solver_B, qs_Coral_P2, dwt2d, qs_Coral_P1
	Q6	bt_solver_C, huffman*, lavaMD, sp_solver_B, gaussian*, randomaccess, lud_C*, stream, cfd*, qs_NoFission*, needle*, qs_Coral_P1

US-dominant (CIx3, MIx3, USx6)	Q7	heartwall*, hotspot*, hotspot3D, gaussian*, stream, lud_B, pathfinder, qs_NoFission*, qs_Coral_P2, backprop*, qs_NoCollisions, dwt2d
	Q8	bt_solver_C, hotspot3D, lavaMD, stream, cfd*, lud_B, qs_Coral_P1, needle*, kmeans, qs_Coral_P2, qs_NoFission*, qs_NoCollisions
	Q9	lavaMD, hotspot3D, hotspot*, sp_solver_B, lud_C*, randomaccess, qs_Coral_P1, dwt2d, kmeans, needle*, qs_NoCollisions, qs_Coral_P2
Balanced (CIx4, MIx4, USx4)	Q10	lavaMD, huffman*, hotspot3D, bt_solver_C, lud_C*, lud_B, stream, sp_solver_C, qs_NoCollisions, needle*, pathfinder, qs_Coral_P1
	Q11	huffman*, hotspot3D, hotspot*, bt_solver_B, cfd*, lud_C*, stream, gaussian*, qs_Coral_P2, needle*, pathfinder, dwt2d
	Q12	lavaMD, hotspot*, huffman*, heartwall*, sp_solver_C, lud_C*, randomaccess, gaussian*, needle*, pathfinder, qs_NoCollisions, backprop*

Machines/platform: For our single-node evaluation, we utilize an A100-based heterogeneous server. The details of our platform setups are as follows:

GPU	NVIDIA A100 40GB PCIe 250W TDP
Operating System	Ubuntu 20.04.4 LTS, Kernel Version: 5.4.0-137-generic
Software	CUDA Version: 11.6, Driver Version: 510.108.03, Python Version: 2.7.18, Slurm Version: 24.08.0

Note that our approach can als run on a cluster given that root access is granted

Baselines: In our evaluation, specifically in the second scenario, we compare our approach to the conventional exclusive scheduling method that does not involve partitioning node GPU resources. Additionally, we evaluate our ML-based approach against several existing naive scheduling and partitioning approaches. The following methods are specifically compared to our approach:

- **Time Sharing (Baseline):** Jobs in the given job mix are executed using the entire GPU resources exclusively without co-scheduling/partitioning.
- **MIG Only (C = 2):** Following our previous study [Arima+ICPPW'22], we test a MIG only option with the concurrency C at 2. The job set selections and assignments are optimal, i.e., exhaustively chosen from all the possible setups.
- **MPS Only (C ≤ Cmax):** We test the MPS only option with concurrency selections (C ≤ Cmax). The job set selections and resource assignments are determined through an exhaustive search.
- **MIG+MPS Default (C ≤ Cmax):** The MIG partitioning is selected so that the average throughput across Q1-Q12 is maximized. The MPS allocation is set to the default mode. The job set selections (LJS) are optimal, i.e., they are chosen through an exhaustive search within the designated concurrency limit and configuration space.

- **MIG+MPS w/ RL (C ≤ Cmax):** Our proposed reinforcement learning-based co-optimization of co-scheduling and hierarchical partitioning.

Metrics: Throughput, application performance, fairness

Evaluation scenario 1: Offering hierarchical partitioning functionality on RJMS
Suppose we have GPU jobs submitted to a dedicated co-scheduling queue, and we assign them to GPUs while accepting oversubscribing them. To this end, we co-optimize the job pair selections from the queue as well as the GPU partitioning setups. In particular, we partition the GPUs in a hierarchical manner using different granular partitioning knobs (e.g., MPS and MIG). In this evaluation, we test the functionality of the hierarchical GPU partitioning we newly augmented to the RJMS.

Evaluation scenario 2: Throughput improvement by ML-based co-scheduling and partitioning optimization
We then demonstrate the throughput improvement by using our RL-based co-scheduling pair selections and GPU partitioning methodology. This sophistication works as a user-level meta scheduler that can interact with the above customized RJMS using the srun/sbatch command. The detailed setups of the RL agent, reward function, neural network, etc. are identical to the following paper: Urvij Saroliya, et al. "Hierarchical Resource Partitioning on Modern GPUs: A Reinforcement Learning Approach" In *CLUSTER*, pp.185-196 (2023)

Experimental results (scenario1): We first perform various functionality evaluations to check if our hierarchical GPU partitioning works. We check the following perspectives here: (1) coarse-grained GPU partitioning functionality (via MIG feature) implemented in our prolog script; (2) fine-grained GPU partitioning feature (MPS) works inside of each MIG partition; (3) GPU co-scheduling tests work over these two features.

Coarse-Grained GPU Partitioning via the MIG Feature:
We configured Slurm configuration files using the GRES plugin to enable resource-wise job scheduling, and each compute node is configured as follows:

```
$ sinfo -h -N $partition $hostlist $statelist -o "%N %P %C %O %m %e
%t %Z %G"
localhost debug* 0/64/0/64 3.06 1 18898 idle 2
gpu:t1000:1(S:0),gpu:a100_7g.40gb:4(S:0),gpu:a100_4g.20gb:2,gpu:a10
0_3g.20gb:2
```

(a) “gpu:a100_7g.40gb:4” is one big MIG compute instance that occupies the whole GPU.
(b) “gpu:a100_4g.20gb:2” is a MIG compute instance that occupies 4 out of 7 GPU nodes and 4 out of 8 memory modules.
(c) “gpu:a100_3g.20gb:2” is a MIG compute instance that occupies 3 out of 7 GPU nodes and 4 out of 8 memory modules.

Note (b) and (c) can co-exist on the GPU at the same time, while they cannot be located while (a) is being used. We assume the user designates one of them to launch a job using “--gres=gpu:a100_X:Y” option where X is chosen from “a100_7g.40gb”, “a100_4g.20gb”, or “a100_3g.20gb” while Y is set to 1-4 for “gpu:a100_7g.40gb:4” but 1-2 for the others. Y is the number of job slots associated with the given MIG GPU partition, and if the partition runs out

of slots, then no job can be assigned to the partition any further. This concurrent job execution within a MIG partition is realized by use of the MPS-based fine-grained partitioning.

We submit an interactive job that executes the “nvidia-smi -L” command that lists the available GPU(s) for the job, which depends on the selected partitioning option. As shown below, only the requested GPU resource is assigned to the job. Note our custom prolog script internally handles these dynamic GPU partitioning modifications.

```
$ srun --gres=gpu:a100_3g.20gb:1 nvidia-smi -L
GPU 0: NVIDIA A100-PCIE-40GB (UUID:
GPU-5b6ec351-6067-def5-6148-17bbe9ac6a64)
  MIG 3g.20gb   Device  0: (UUID:
MIG-06a42a19-4578-5d99-977c-2bcfeee1d876)

$ srun --gres=gpu:a100_4g.20gb:1 nvidia-smi -L
GPU 0: NVIDIA A100-PCIE-40GB (UUID:
GPU-5b6ec351-6067-def5-6148-17bbe9ac6a64)
  MIG 4g.20gb   Device  0: (UUID:
MIG-9c8663c6-ffce-5255-9de8-1a7bfaeba242)

$ srun --gres=gpu:a100_7g.40gb:1 nvidia-smi -L
GPU 0: NVIDIA A100-PCIE-40GB (UUID:
GPU-5b6ec351-6067-def5-6148-17bbe9ac6a64)
  MIG 7g.40gb   Device  0: (UUID:
MIG-d647ab52-bb82-5952-90b5-7dd73e4154dc)
```

We then co-schedule multiple GPU jobs at the same time on the GPU with several different partitioning options. We observed the following outputs. The scheduler assigned only the requested resources in all cases. Note, in the last two cases, the jobs are executed sequentially as one of the co-scheduled jobs occupies the entire GPU.

```
$ srun --gres=gpu:a100_4g.20gb:1 nvidia-smi -L & srun
--gres=gpu:a100_3g.20gb:1 nvidia-smi -L &
[1] 376088
[2] 376089
GPU 0: NVIDIA A100-PCIE-40GB (UUID:
GPU-5b6ec351-6067-def5-6148-17bbe9ac6a64)
  MIG 3g.20gb     Device  0: (UUID:
MIG-06a42a19-4578-5d99-977c-2bcfeee1d876)
GPU 0: NVIDIA A100-PCIE-40GB (UUID:
GPU-5b6ec351-6067-def5-6148-17bbe9ac6a64)
  MIG 4g.20gb     Device  0: (UUID:
MIG-9c8663c6-ffce-5255-9de8-1a7bfaeba242)

[1]-  Done                    srun --gres=gpu:a100_4g.20gb:1 nvidia-smi -L
[2]+  Done                    srun --gres=gpu:a100_3g.20gb:1 nvidia-smi -L

$ srun --gres=gpu:a100_4g.20gb:1 nvidia-smi -L & srun
--gres=gpu:a100_7g.40gb:1 nvidia-smi -L &
```

```
[1] 376576
[2] 376577
GPU 0: NVIDIA A100-PCIE-40GB (UUID:
GPU-5b6ec351-6067-def5-6148-17bbe9ac6a64)
  MIG 4g.20gb   Device  0: (UUID:
MIG-9c8663c6-ffce-5255-9de8-1a7bfaeba242)
GPU 0: NVIDIA A100-PCIE-40GB (UUID:
GPU-5b6ec351-6067-def5-6148-17bbe9ac6a64)
  MIG 7g.40gb   Device  0: (UUID:
MIG-d647ab52-bb82-5952-90b5-7dd73e4154dc)
```

[1]- Done srun --gres=gpu:a100_4g.20gb:1 nvidia-smi -L
[2]+ Done srun --gres=gpu:a100_7g.40gb:1 nvidia-smi -L

```
$ srun --gres=gpu:a100_3g.20gb:1 nvidia-smi -L & srun
--gres=gpu:a100_7g.40gb:1 nvidia-smi -L &
[1] 376864
[2] 376865
GPU 0: NVIDIA A100-PCIE-40GB (UUID:
GPU-5b6ec351-6067-def5-6148-17bbe9ac6a64)
  MIG 7g.40gb   Device  0: (UUID:
MIG-d647ab52-bb82-5952-90b5-7dd73e4154dc)
GPU 0: NVIDIA A100-PCIE-40GB (UUID:
GPU-5b6ec351-6067-def5-6148-17bbe9ac6a64)
  MIG 3g.20gb   Device  0: (UUID:
MIG-06a42a19-4578-5d99-977c-2bcfeee1d876)
```

[1]- Done srun --gres=gpu:a100_3g.20gb:1 nvidia-smi -L
[2]+ Done srun --gres=gpu:a100_7g.40gb:1 nvidia-smi -L

Fine-Grained GPU Partitioning via the MPS Feature

On each MIG partition, we further partition it using a fine-grained resource partitioning feature (MPS). The users can request part of the GPU resource within the partition, designated by the percentage. More specifically, when one submits a job to the “gpu:a100_7g.40gb:1” partition and requests 50% of the compute resource within the partition, then he/she puts the following options: “ --gres=gpu:a100_7g.40gb:1” and “--export=All,MPS_PERCENTAGE=50”. Here, we measured the performance of 4 basic streaming benchmarks (copy, scale, add, and triad), while changing the MPS percentage, and obtained the following outputs:

```
$ srun --gres=gpu:a100_7g.40gb:1 --export=All,MPS_PERCENTAGE=100
./cuda-stream/stream -i 400
```

Function	Rate (GiB/s)	Avg time(s)	Min time(s)	Max time(s)
Copy:	1283.8396	0.00078101	0.00077891	0.00078583
Scale:	1279.1412	0.00078372	0.00078177	0.00079179
Add:	1273.5741	0.00118441	0.00117779	0.00119114
Triad:	1274.6062	0.00118443	0.00117683	0.00119805

```
$ srun --gres=gpu:a100_7g.40gb:1 --export=All,MPS_PERCENTAGE=50
./cuda-stream/stream -i 400
```

```
Function      Rate (GiB/s)  Avg time(s)  Min time(s)  Max time(s)
-------------------------------------------------------------
Copy:         1206.2997     0.00083065   0.00082898   0.00083494
Scale:        1203.5306     0.00083258   0.00083089   0.00084090
Add:          1251.0352     0.00120544   0.00119901   0.00121093
Triad:        1252.0310     0.00120284   0.00119805   0.00122094
```

```
$ srun --gres=gpu:a100_7g.40gb:1 --export=All,MPS_PERCENTAGE=10
./cuda-stream/stream -i 400
```

```
Function      Rate (GiB/s)  Avg time(s)  Min time(s)  Max time(s)
-------------------------------------------------------------
Copy:         252.3345      0.00397639   0.00396299   0.00410700
Scale:        251.4420      0.00398745   0.00397706   0.00411701
Add:          357.3067      0.00420556   0.00419807   0.00433803
Triad:        359.7996      0.00417673   0.00416899   0.00431180
```

When we set the percentage to 50, the performance degradation is almost negligible. However, we observed a significant performance drop when we set it to 10. This is a typical behavior of memory-bound applications – scaling down computational throughput (or Flop/s) does not affect the application performance when the arithmetic intensity is relatively small compared with the F/B rate of the machine.

We present in Figure 7.10 the memory bandwidth of these basic benchmarks as a function of the MPS percentage for these 3 different MIG partitioning options. For each of these partitions, we scaled the MPS percentage, and their scalability are presented in the figures below.

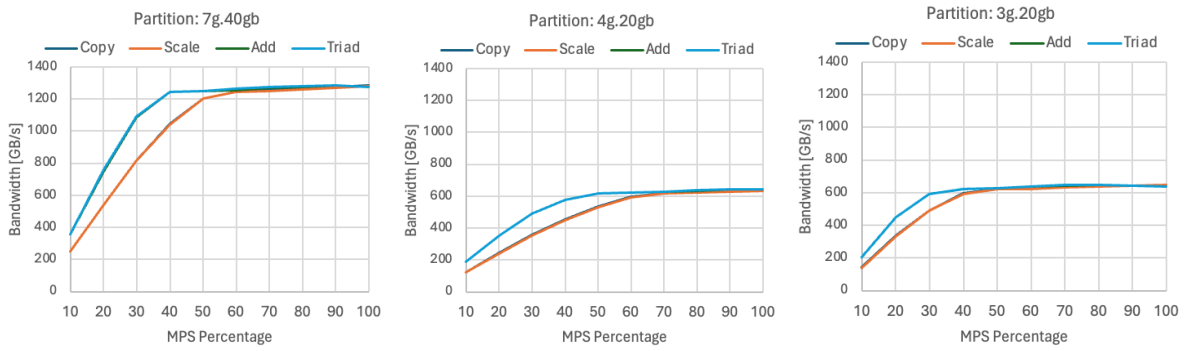


Figure 7.10: Measured Memory Bandwidth as a Function of MPS Percentage

Hierarchical GPU Partitioning via the MIG and MPS Features

We then co-schedule multiple jobs with multiple different resource request options, and observe the behavior, especially when more jobs than the available partitioning slots are submitted. As the first example, we submitted 6 jobs on the “gpu:a100_7g.40gb:4” partition, each of which requests one job slot on the partition. We used the streaming benchmark here

and observed the following behavior. As this partition can execute up to 4 jobs concurrently, the last two jobs were stacked in the queue, and were launched right after the requested resources became available.

```
$ srun --gres=gpu:a100_7g.40gb:1 --export=All,MPS_PERCENTAGE=25
./cuda-stream/stream -i 400 & srun --gres=gpu:a100_7g.40gb:1
--export=All,MPS_PERCENTAGE=25 ./cuda-stream/stream -i 400 &
[1] 395124
[2] 395125
$ srun --gres=gpu:a100_7g.40gb:1 --export=All,MPS_PERCENTAGE=25
./cuda-stream/stream -i 400 & srun --gres=gpu:a100_7g.40gb:1
--export=All,MPS_PERCENTAGE=25 ./cuda-stream/stream -i 400 &
[3] 395213
[4] 395214
$ srun --gres=gpu:a100_7g.40gb:1 --export=All,MPS_PERCENTAGE=25
./cuda-stream/stream -i 400 & srun --gres=gpu:a100_7g.40gb:1
--export=All,MPS_PERCENTAGE=25 ./cuda-stream/stream -i 400 &
[5] 395293
[6] 395294

srun: job 1110 queued and waiting for resources
srun: job 1111 queued and waiting for resources

 STREAM Benchmark implementation in CUDA
 Array size (double precision) = 536.87 MB
 using 192 threads per block, 349526 blocks
 output in IEC units (KiB = 1024 B)

Function      Rate (GiB/s)  Avg time(s)  Min time(s)  Max time(s)
-------------------------------------------------------------
Copy:         706.2307      0.00273704   0.00141597   0.00366116
Scale:        702.6812      0.00277921   0.00142312   0.00361609
Add:          951.6648      0.00337473   0.00157619   0.00500917
Triad:        965.2433      0.00332884   0.00155401   0.00444293
srun: job 1110 has been allocated resources
 STREAM Benchmark implementation in CUDA
 Array size (double precision) = 536.87 MB
 using 192 threads per block, 349526 blocks
 output in IEC units (KiB = 1024 B)

Function      Rate (GiB/s)  Avg time(s)  Min time(s)  Max time(s)
-------------------------------------------------------------
Copy:         523.5681      0.00305376   0.00190997   0.00387907
Scale:        581.4117      0.00287126   0.00171995   0.00376487
Add:          722.2427      0.00329247   0.00207686   0.00408602
Triad:        642.1164      0.00353842   0.00233603   0.00469303
srun: job 1111 has been allocated resources
 STREAM Benchmark implementation in CUDA
 Array size (double precision) = 536.87 MB
```

```
 using 192 threads per block, 349526 blocks
 output in IEC units (KiB = 1024 B)

Function        Rate (GiB/s)  Avg time(s)  Min time(s)  Max time(s)
-------------------------------------------------------------
Copy:           598.7586      0.00290515   0.00167012   0.00378084
Scale:          548.5619      0.00307704   0.00182295   0.00388598
Add:            627.0762      0.00361930   0.00239205   0.00483990
Triad:          688.0420      0.00341054   0.00218010   0.00436807
```

In the second example, we submitted 3 jobs on the “gpu:a100_4g.20gb:2” partition and 3 jobs on the “gpu:a100_3g.20gb:2”, each of which requested one job slot. We used the streaming benchmark here and observed the following behavior. As each of these partitions can execute up to two jobs concurrently, the last two jobs were stacked in the queue, and were launched right after the requested resources became available.

```
$ srun --gres=gpu:a100_4g.20gb:1 --export=All,MPS_PERCENTAGE=50
./cuda-stream/stream -i 400 & srun --gres=gpu:a100_3g.20gb:1
--export=All,MPS_PERCENTAGE=50 ./cuda-stream/stream -i 400 &
[1] 397333
[2] 397334
$ srun --gres=gpu:a100_4g.20gb:1 --export=All,MPS_PERCENTAGE=50
./cuda-stream/stream -i 400 & srun --gres=gpu:a100_3g.20gb:1
--export=All,MPS_PERCENTAGE=50 ./cuda-stream/stream -i 400 &
[3] 397434
[4] 397435
$ srun --gres=gpu:a100_4g.20gb:1 --export=All,MPS_PERCENTAGE=50
./cuda-stream/stream -i 400 & srun --gres=gpu:a100_3g.20gb:1
--export=All,MPS_PERCENTAGE=50 ./cuda-stream/stream -i 400 &
[5] 397564
[6] 397565
srun: job 1124 queued and waiting for resources
srun: job 1125 queued and waiting for resources
 STREAM Benchmark implementation in CUDA
 Array size (double precision) = 536.87 MB
 using 192 threads per block, 349526 blocks
 output in IEC units (KiB = 1024 B)

Function        Rate (GiB/s)  Avg time(s)  Min time(s)  Max time(s)
-------------------------------------------------------------
Copy:           622.3003      0.00206568   0.00160694   0.00309706
Scale:          620.7346      0.00220128   0.00161099   0.00362015
Add:            628.9569      0.00309450   0.00238490   0.00468302
Triad:          629.7123      0.00294572   0.00238204   0.00439286
srun: job 1124 has been allocated resources
 STREAM Benchmark implementation in CUDA
 Array size (double precision) = 536.87 MB
 using 192 threads per block, 349526 blocks
 output in IEC units (KiB = 1024 B)
```

```
Function        Rate (GiB/s)  Avg time(s)  Min time(s)  Max time(s)
-------------------------------------------------------------------
Copy:           536.4932      0.00263514   0.00186396   0.00364900
Scale:          532.2045      0.00277924   0.00187898   0.00366592
Add:            616.5676      0.00360849   0.00243282   0.00463605
Triad:          615.9640      0.00344774   0.00243521   0.00463104
srun: job 1125 has been allocated resources
 STREAM Benchmark implementation in CUDA
 Array size (double precision) = 536.87 MB
 using 192 threads per block, 349526 blocks
 output in IEC units (KiB = 1024 B)

Function        Rate (GiB/s)  Avg time(s)  Min time(s)  Max time(s)
-------------------------------------------------------------------
Copy:           534.5105      0.00299328   0.00187087   0.00367212
Scale:          531.9346      0.00285389   0.00187993   0.00365090
Add:            618.0819      0.00378833   0.00242686   0.00478697
Triad:          615.7831      0.00396609   0.00243592   0.00500703
                                    ⋮
```

Experimental results (scenario2): Figure 7.11 compares throughput among different methods and across different workloads. The horizontal axis represents executed workloads (AM: Arithmetic Mean), while the vertical axis indicates relative throughput normalized to that of Time Sharing for each workload. In general, the proposed reinforcement learning-based approach outperforms all the other methods for almost all the workloads. Compared with the Time Sharing, it achieves ***1.516 or 1.873*** times throughput improvement on average or at best, respectively.

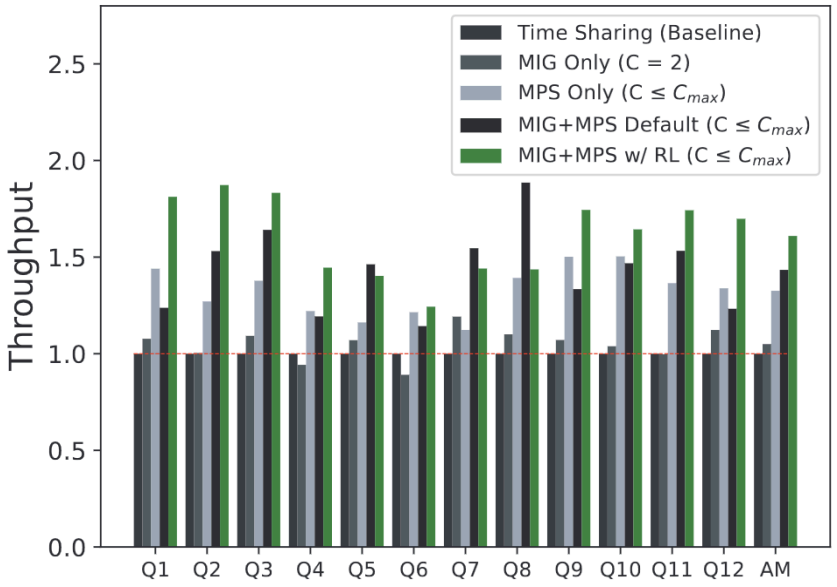


Figure 7.11: Throughput Comparison

Figure 7.12 presents the average throughput as a function of the window size (or number of jobs within the scheduling target). The vertical axis represents the average throughput based

on all of the 12 job queues, and the horizontal axis shows the window size (or the number of jobs to be scheduled from a given queue, the first W jobs from the head are chosen). As shown in the figures, the throughput increases as we scale the window size because our approach can find better co-scheduling groups when the window size is higher.

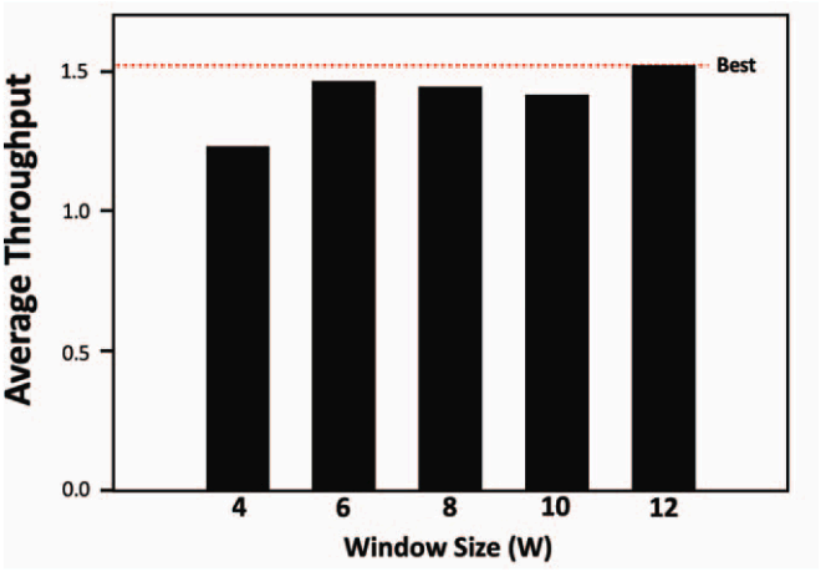


Figure 7.12 Average Throughput Comparison for various Window Sizes

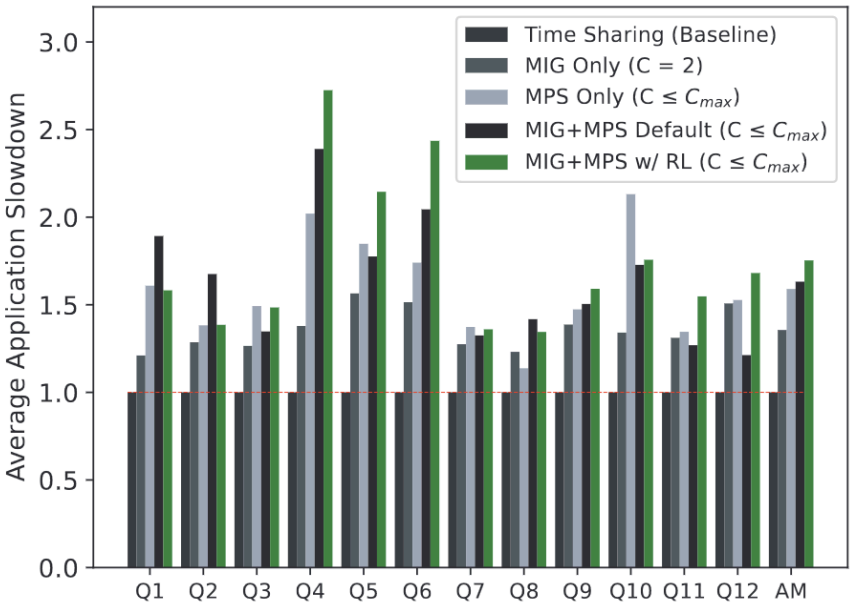


Figure 7.13: Per Application Slowdown

Figure 7.13 demonstrates the average application slowdown caused by co-scheduling for different methods across different job queues. The X-axis lists evaluated workloads, while the Y-axis represents the average application slowdown. We define the application slowdown (AppSlowdown) for a given job taken from the given queue (J ∈ Qi) as follows:

$$AppSlowdown(J) = \frac{CoRunAppTime(J)}{SoloRunAppTime(J)}$$

Here, CoRunAppTime(J) or SoloRunAppTime(J) denote the space-sharing execution time or the solo-run execution time for the given job (J), respectively. We calculate the average across all the jobs in the given queue for each method. The average application slowdown for our approach is on average 1.829 and is 1.345 at best. As co-scheduling can offer more concurrency (4 in our approach), it can achieve higher throughput in total as introduced before. As our approach can trade-off the application slowdowns and concurrency in a better way, it achieves higher total system throughput as a consequence compared with others.

Figure 7.14 compares the fairness in scheduling among different methods across different workloads. We utilize the following fairness metric for the given queue (Qi):

$$Fairness(Q_i) = \frac{\min_{J \in Q_i}(AppSlowdown(J))}{\max_{J \in Q_i}(AppSlowdown(J))}$$

A higher value is better for this metric, and the highest one is 1. More specifically, when this fairness metric is equal to 1, the maximum slowdown becomes exactly the same as the minimum slowdown, which means all the applications suffer from the same degree of slowdown. According to the figure, ours is comparable in fairness with the other approaches except for the Time Sharing, even though ours outperforms them in throughput. Note we can improve the fairness in our approach by taking it into account in the reward function.

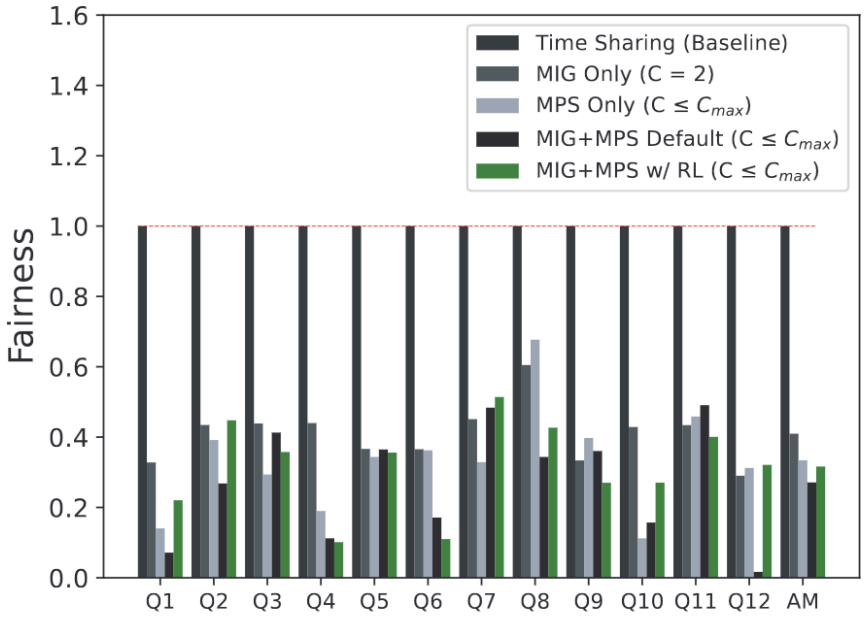


Figure 7.14: Fairness Comparison

7.3 Power-aware scheduling

In this section we report on our investigations on scheduling applications by the resource and job manager, when power constraints are applied. Thus, the main objective is first to stay within the power limits at all time, and then to optimize the throughput or the energy consumption while under varying power caps.

Scheduling with power predictions

We initially started to investigate augmenting the usual scheduling policies such as EASY backfilling with power information to optimize scheduling during power constrained periods

(power capping). As users are not necessarily experts in the power consumption of their jobs, we had to rely on power predictions for the submitted tasks, to decide on their placement before their execution. The simulations presented here use a sampling of the Marconi platform execution traces, in order to have accurate power measurement of the real execution.

Power estimations

To avoid replicating work done by partners in the power estimations, we decided to start with simpler estimations not relying on machine learning. This is of course much coarser as an estimation as the predictions obtained in the other approaches, specifically for the machine learning aspects. However, we will see that the results are already really encouraging and demonstrate the soundness of the approach. In the following figures, we will present results with three basic estimations:

- Naive: the power estimation is set at the maximum of the possible consumption of the job, according to the number of resources requested. This serves as a baseline for the current state of practice.
- Max: the power estimation is set with an average of max consumption of jobs submitted by the same user. This weighted average is done over a sliding time window, with a decreasing weight for older jobs.
- Average: this power estimation is done with the same approach as Max, but relies on the average jobs power consumption.

Five different power capping scenarios are presented in Figure 7.15, with available power from 10% to 70% of the dynamic power available (that is, the power range consumed above the minimal idle power). For Max and Mean power prediction we present two boxplots, the left one being the predicted value for energy consumption before the execution of the schedule, and the right one being the real energy consumption after execution of the applications.

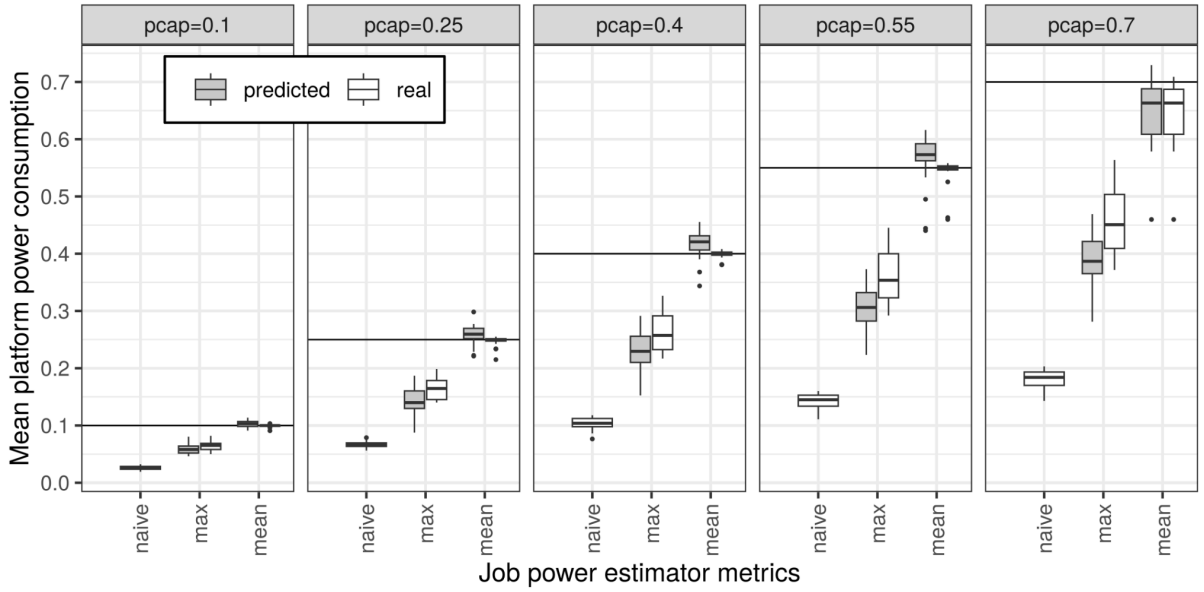


Figure 7.15: Mean power consumption during power capped periods (from 10% to 70% of total power).

As can be seen on the figure, the predictions for Max and Mean are reasonably close to the real measured power consumption during execution (roughly within 10%). The main

difference is in how the scheduler is able to select and place more tasks when the prediction is lower, as the predictions are always ordered: Naive predicts the full utilization of all the nodes, Max predicts an execution at the maximum power consumption seen on previous jobs by the same user, and finally Mean predicts a consumption at the average power consumption of the preceding jobs. This is reflected in the mean power consumption of the platform. With the Naive prediction, the mean power consumption during the power capping periods is far from the cap, first because applications are not consuming their predicted power, and also because there often is some left over unallocated power as there are no applications fitting perfectly in these leftovers. The Max power prediction limits the power allocated to applications, so this frees some power to allocate more applications in this period. Finally the Mean prediction is a bit too optimistic and sometimes goes over the power caps. This could be resolved by enforcing the cap at the node level for all applications, ensuring that the platform stays within the designated power envelope.

Using more of the available power translates directly to running more jobs during the whole day. Since the workload from Marconi is sampled and simulated over a week-long period with several power capping events, the throughput is only comparable for a fixed power capping scenario. In order to compare the impact of the intensity of power capping we present in Figure 7.16 another metric of improved job execution, namely the mean turnaround time increase. This metric allows us to see for different power capping values, how the jobs are delayed on average in their execution. As you can see in this figure, when 70% of the dynamic power is available the mean prediction allows for an execution with almost no turnaround time increase. This is partly explained by the fact that a supercomputer rarely runs at 100% of its peak available power, and so there are enough applications running at 70% or less to fill the power capping periods without a global impact on the schedule.

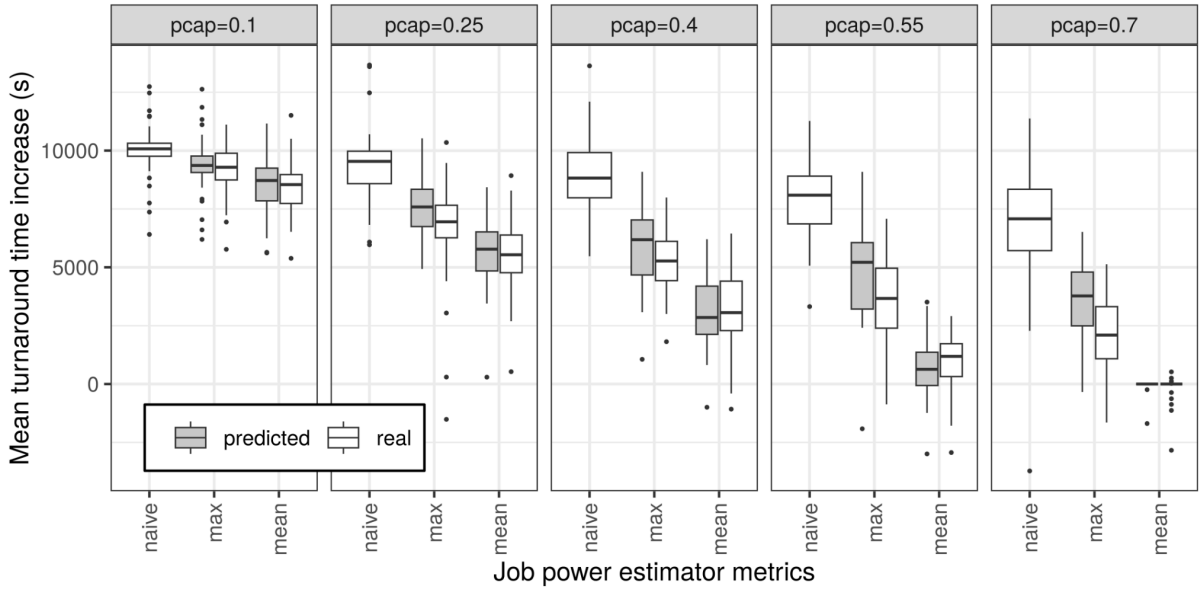


Figure 7.16: Mean turnaround time increase

For the Naive power prediction however, the impact is immediately visible as part of the platform will be kept artificially idle during the power cap periods. Finally, with large power cap settings the gap between the different solutions is less important as there is not a lot of energy to use anyway.

Scheduling with Eco mode (DVFS volunteers)

To go beyond this first power estimation approach, we investigated giving some freedom to the users, by allowing them to pick the execution mode of their jobs among two possible mode: a normal mode where executions are left as they normally are, and an Eco-mode where jobs can be individually capped if needed to guarantee the global power cap. In order to assess this Eco-mode without interference from power aware Easy-backfilling, we used in this approach a classical power oblivious scheduling heuristic, using the application power capping only if required by the power usage at the start of a power constrained period. So if there are no power capping periods, our Eco-mode is not taken into account at all.

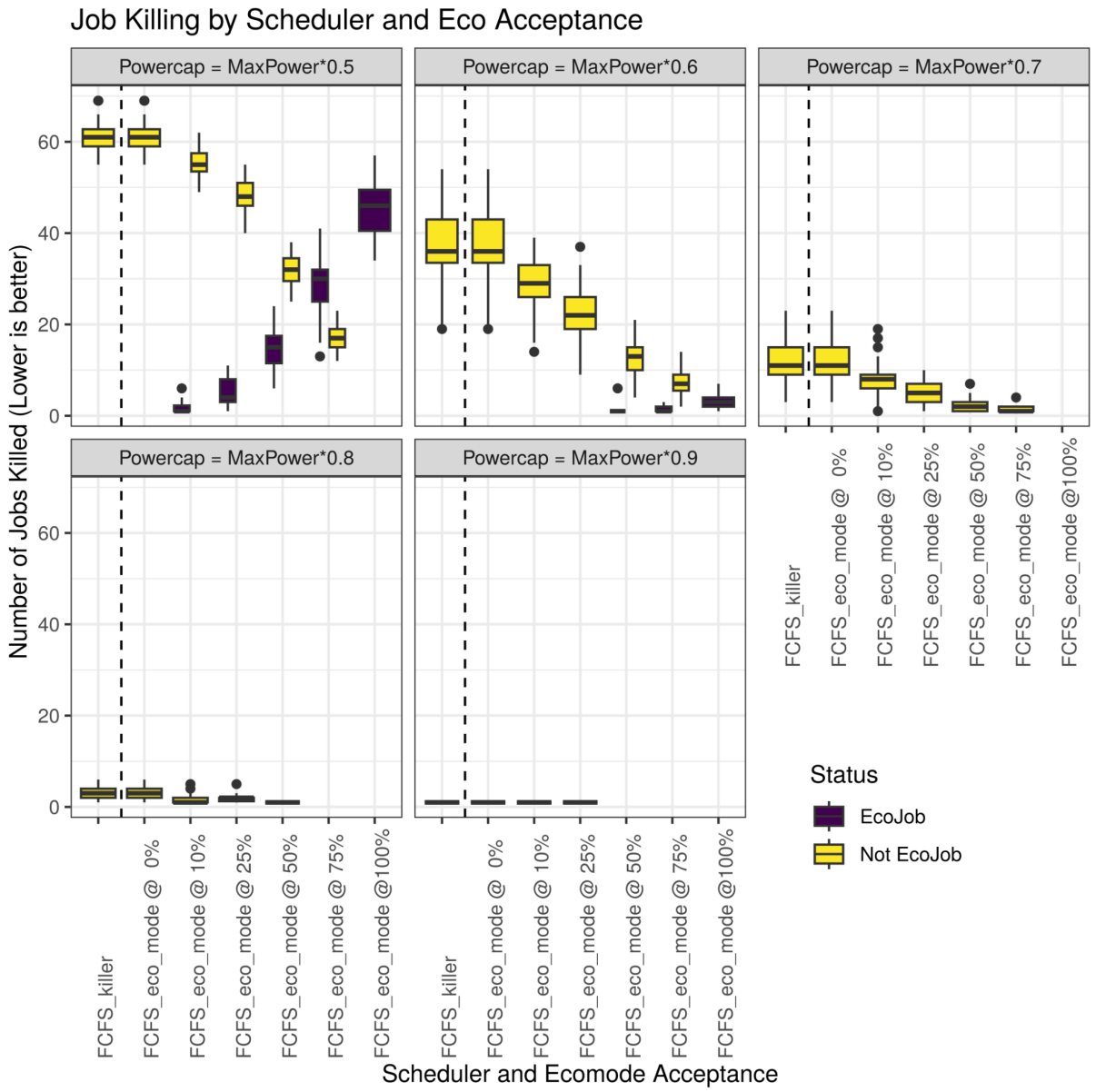


Figure 7.17: Jobs killed per volunteer percentage for different power caps.

The Eco-mode approach was presented extensively in deliverable 2.3, and will only be briefly recalled here. When a power capping period starts, all the Eco-mode jobs currently running are iteratively changed to lower CPU frequencies until the power cap is satisfied. In the case where there are not enough volunteers currently in execution, or if the power cap is extremely demanding, some applications will be killed until the power cap is respected. During the

power capping period, whenever a job ends and frees some power capacity, the Eco-mode jobs previously slowed down are set to higher frequencies to use that newly available power. If all the Eco-mode jobs are at their optimal frequency and there is some power left, a new job can be started as long as its power consumption fits the available power without impacting currently running jobs. Real power measured on the platform is used to maximize the power consumption during power capping periods. As shown in Figure 7.17, in cases where the power cap is high, having enough volunteers is enough to guarantee that the transition is smooth without killing any jobs. When the power cap gets more restrictive, or if there are not enough volunteers, the number of jobs increases notably. To motivate users to declare their jobs as Eco-mode, non Eco-mode jobs are selected first to be killed, until only Eco-mode remains. This is visible in the scenario where only 50% of max power is available. Note that this time we are not considering the dynamic power, but the total power (thus including idle power). The motivation for this difference in power considered is to be able to put some nodes in deep sleep or total shutdown if this helps to avoid killing some jobs. We did not however reach that step yet in our experiments.

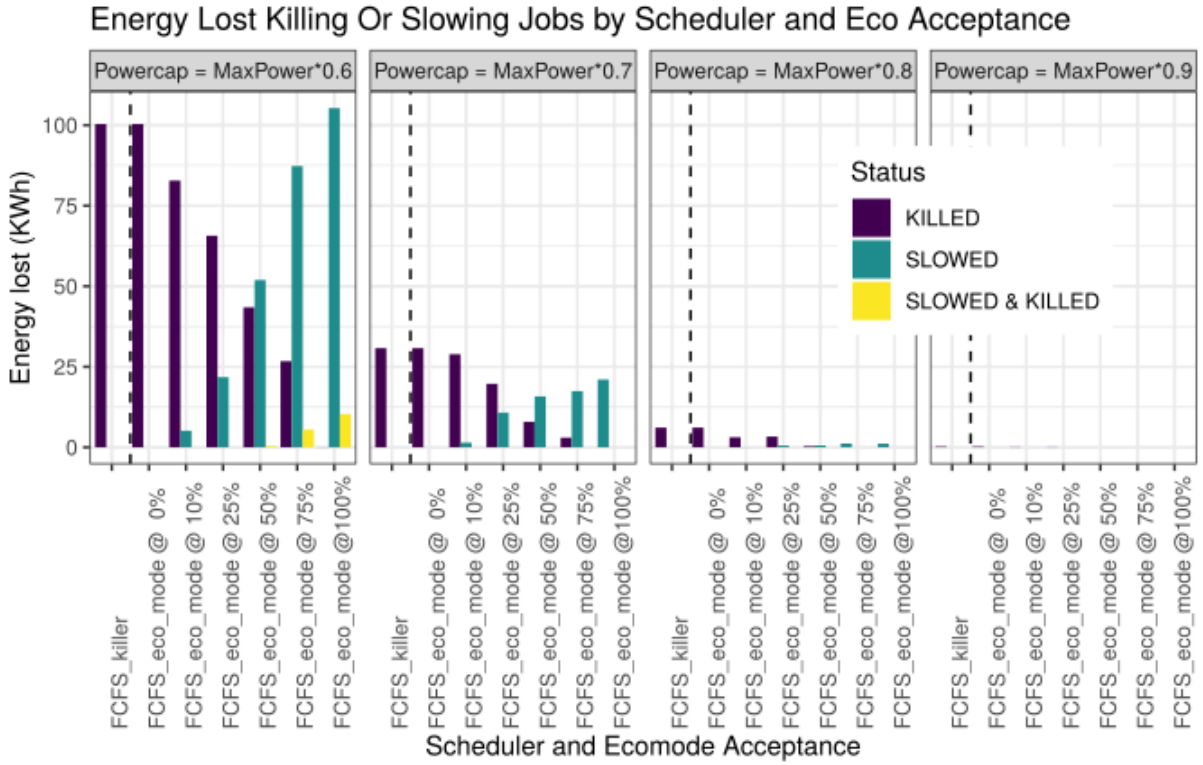


Figure 7.18: Lost energy compared to a normal execution

In Figure 7.18, we show how much energy is lost by slowing down or killing applications. Indeed, whenever an application is slowed down, its total energy consumed during the execution is more than during a normal execution, as the time increase is energy-wise more important than the power reduction (since a significant part of the power consumption is simply the base power consumption). Similarly, killed jobs used some energy without producing any result, thus that energy is lost. For more details on these experiments, readers are invited to check our research report[7] available online.

[7] Luc Angelelli, Danilo Carastan-Santos, Pierre-François Dutot. Run your HPC jobs in Eco-Mode: revealing the potential of user-assisted power capping in supercomputing systems. 2024. ⟨hal-04525291⟩ https://hal.science/hal-04525291v1

7.4 Power/thermal control

In the context of power/thermal control at UNIBO, our activities can be divided into three parts: (i) ControlPULP: an open-source, low-level power controller. (ii) A room-level thermal anomaly prediction framework that introduces a statistical rule-based approach for thermal anomaly detection and machine learning tools for thermal anomaly prediction. (iii) The integration of machine learning models into the production system, provides a framework for implementing the machine learning model in production.

ControlPULP

The goal of the ControlPULP is to provide an HW/SW open-source low-level power controller implementation to be integrated in HPC System on Chip (SoC) design, orchestrating the internal and external physical power management interfaces (eg. VRM, PLL) to support power capping and thermal capping services while implementing standard power management interfaces with the O.S..

Being an open-source IP its evaluation depends on the actual implementation into a real HPC SoC. The ControlPULP IP has been developed within the European-Processor-Initiative and it is expected to be integrated in the Rhea processor by SiPEARL. Within the context of the REGALE project the power management firmware and its interface toward the node controller and job controller has been evaluated. This evaluation has thus been executed on an emulation platform consisting of an Xilinx Ultrascale+ Zynq FPGA. The chosen FPGA consists of two partitions, one composed by a core complex realized on silicon (PS) and the other by a programmable logic fabric (PL). The PS part consists of dual ARM cores booting Linux, which are used to simulate an HPC workload (performance model), the HPC SoC thermal model and power model, as well as to implement real O.S. power management interfaces. The PL part emulates the ControlPULP HW/SW IP by means of the synthesized design and power control firmware executed into the programmable cores constituting the ControlPULP HW IP. Moreover the ControlPULP design has been extended with HW mailbox and shared memory regions to emulate the HW support needed in HPC systems for the exchange of power management commands and information between the application cores and the power controller unit, usually referred as Power Management Interfaces (PMI). The results obtained by the sophistication added in the regale project to the ControlPULP design focused on the firmware aspect, with specific focus on the O.S. to low-level power management services, like power capping and thermal capping. Indeed the REGALE HPC powerstack components (Job Manager, Node Manager, System Power Manager) focused on the other half of the power management problem consisting of the chain of interfaces and management control between HPC application to the O.S. power management and power monitoring interface - in Linux O.S. systems referred to as Governors. In HPC SoCs based on the ARM ISA, the default governor is named ARM SCMI, as it adheres to the System Control and Management Interface (SCMI) standard. We extended the ControlPULP FPGA emulation platform to support the SCMI standard and evaluated how latencies in the power management command propagation across the different stack of controllers impairs the power management functionality. Then we provided a sophistication in the ControlPULP firmware which has been found to be a bottleneck in reacting to fast changes in the power management settings due to its internal control loop structure and timing.

Key Performance Indicator

The experimental analysis focuses on the Strategic Objective 1 (SO1) and more specifically on the SO1.3: Minimization of performance degradation under the operation with power constraints. The chosen KPI is performance under thermal and power cap.

Experimental platform

The experimental platform consists of an emulated many-core system composed of 16 tiles of four cores. An application consisting of two phases one memory bound and one cpu bound is simulated. A node manager controller is also simulated to request frequency scaling synchronously with the phase change using SCMI O.S. governor interface to the ControlPULP firmware. The system is co-simulated in an Hardware-in-the-Loop (HIL) setting using the Xilinx Zynq Ultrascale+ ZCU102 ControlPULP emulation system.

Evaluation protocol

We measured the relative performance gain obtained by introducing the sophistication in the ControlPULP firmware to allow instantaneous changes of the internal power management settings as soon as demanded by the SCMI O.S. interface if the requested power management setting reduces the power consumption, if not the setting has the normal latency due to the feedback loop update time which causes an up to 1ms delay.

Results

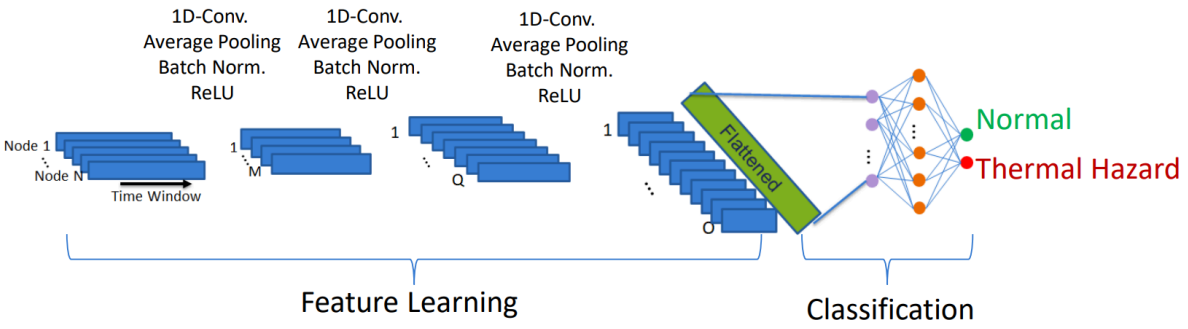
Firmware Configuration	Performance Improvement ideal O.S. PMI	Performance Improvement SCMI PMI
Default ControlPULP firmware	2.75%	2.71%
Improved ControlPULP firmware	3.18%	2.89%

7.5 Room-level thermal anomaly prediction framework

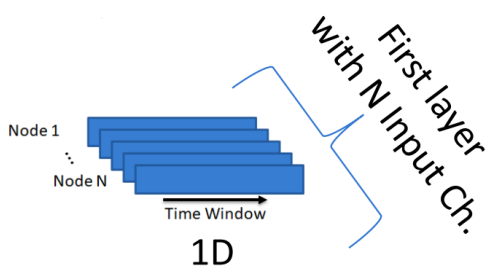
We have developed a room-level thermal anomaly prediction framework capable of detecting and predicting anomalies within HPC clusters. This framework utilizes a statistical rule-based approach derived from the analysis of real reported thermal hazard data to annotate monitoring data. Additionally, it incorporates a flexible machine learning module that can integrate with various classical machine learning and deep neural network (DNN) models. These models are designed to capture the spatiotemporal characteristics of data from computing nodes. Detailed specifications are outlined in D2.3[8].

Figure 7.19 illustrates the briefly different architectures of the TCN models. More details can be found in D2.3 and the HazardNet[9] paper. We evaluated the prediction performance of different AI models using two different test approaches (see Table 7.2). The main distinction between these two approaches is the selection method for the test and training datasets. Our findings indicate that TCN models outperform both LSTM and classical machine learning approaches, prompting further enhancements in their architecture to improve predictive performance.

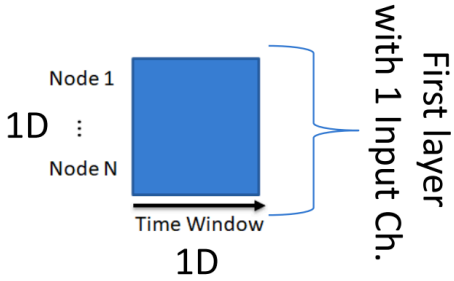
[8] D2.3 Final integration of sophisticated policies in the REGALE prototype
[9] Seyedkazemi Ardebili, M., Acquaviva, A., Benini, L., & Bartolini, A. (2024). HazardNet: A thermal hazard prediction framework for datacenters. Future Generation Computer Systems, 155, 340-353.



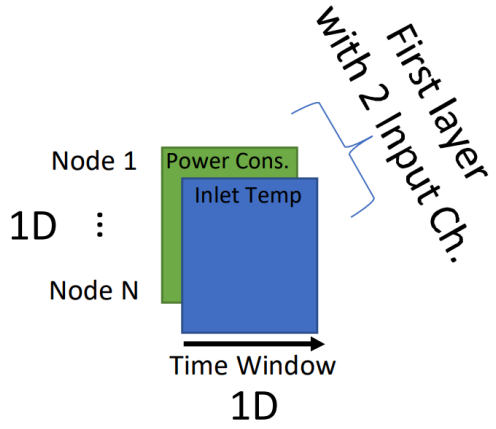
(a) TCN Model with 1DConv. Layers.



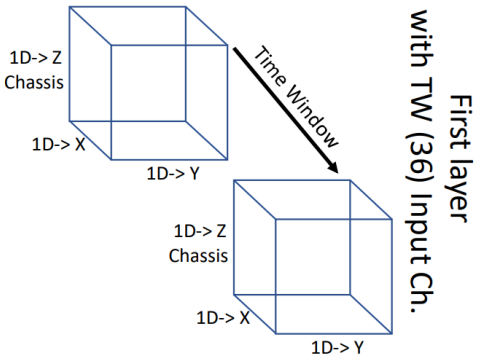
(b) Input Data Structure of the TCN Model with 1DConv. Layers.



(c) Input Data Structure of the TCN Model with 2DConv. Layers.



(d) Input Data Structure of the TCN Model with 2DConv. Layers Power Consumption as a Second Input Channel.



(e) Input Data Structure of the TCN Model with 3DConv. Layers.

Figure 7.19: TCN Model's Architecture and Input Data Structures for Different Types of Convolutional Layers (1DConv., 2DConv., and 3DConv.).

Framework Evaluation

1. Random Test Dataset: In this approach, we randomly selected 20% of the one-year data as the test dataset and trained the models on the remaining 80%. However, we find two concerns about this approach: (i) There is much overlap between each successive sample, meaning that each consecutive sample has a lot of replicated data. As a result, if one of the two consecutive samples is in the training dataset and the other in the test dataset, due to the high overlap of the two samples, the model is indirectly trained by the test sample. (ii) Random selection training and test datasets destroy the chronological order of the training and test samples, which is important for timeseries data because it destroys the causality of the data. i.e., in the test dataset, some samples are chronologically before the training

samples. However, in the objective case implementation, the model is trained with past data to predict the future.

2. Time-separate Test Dataset: To address the issues of the first test approach, in this approach, we simulate a real-case scenario by training the model with data from May and testing the model in the first week of June. We should highlight that using a random selection approach for validating machine learning models, even with time series datasets, is a widely accepted practice in the field. In our study, the dataset was partitioned into sequences of 6-hour periods, where each sample represents a distinct and individual data point. Although there might be similarities between consecutive samples, they are inherently unique. Utilizing random selection for test samples offers several advantages. It enables a comprehensive evaluation of the performance of the model (LVP, SVM, RBF-SVM, SGD-classifier, LSTM, and TCN) across the entire dataset. This approach captures the properties and characteristics of the dataset more effectively than using a time-separated approach. By doing so, it ensures a robust assessment of the model's generalization capabilities, particularly in scenarios where the data distribution may vary over time. We conducted tests using the random test dataset selection approach. However, we were aware of the technical challenges associated with this approach during the implementation in a real system. The actual in-production system should train the model using historical data and utilize it to make predictions for the future. The second test approach offers a perspective on the performance of a model trained on a small portion (1/12) of the dataset, but in a more realistic scenario. By using these two test approaches in conjunction, we can obtain a comprehensive evaluation of both the ML/DL tool selection and the overall performance of the framework.

Table 7.2: Performance of various ML/DNN models in predicting thermal anomalies.

Random Test Dataset								
Exp. No	Size	Model architecture	Input	#Chnls	#Nodes	F1-score	Precision	Recall
Exp.1[19]	–	Last Value Predictor	Inlet Temp.	–	72 Nodes of One Rack	0.72	0.72	0.72
Exp.2[19]	< 1 K	Linear SVM	Inlet Temp.	–	72 Nodes of One Rack	0.55	0.56	0.55
Exp.3[19]	< 1 K	RBF-SVM	Inlet Temp.	–	72 Nodes of One Rack	0.80	0.94	0.86
Exp.4[19]	< 1 K	SGD-classifier	Inlet Temp.	–	72 Nodes of One Rack	0.64	0.76	0.69
Exp.5[19]	8K	LSTM	Inlet Temp.	–	72 Nodes of One Rack	0.84	0.98	0.91
Exp.6[19]	14 K	TCN	Inlet Temp.	72	72 Nodes of One Rack	0.98	0.99	0.98
Time-separate Test Dataset								
Exp. No	Size	Model Architecture	Input	#Chnls	#Nodes	F1-score	Precision	Recall
Exp.7[19]	14 K	1D Conv, Normal	Inlet Temp.	72	72 Nodes of One Rack	0.74	0.7	0.79
Exp.8	14 K	1D Conv, Normal	Inlet Temp.	72	72 Randomly Selected Nodes	0.77	0.66	0.92
Exp.9	3017 K	1D Conv, Normal	Inlet Temp.	3312	3312 All Nodes	0.78	0.66	0.96
Exp.10	1.5 K	2D Conv, Normal	Inlet Temp.	1	72 Randomly Selected Nodes	0.82	0.74	0.93
Exp.11	3 K	2D Conv, Normal	Inlet Temp. & Power	2	72 Randomly Selected Nodes	0.73	0.68	0.8
Exp.12	636 K	2D Conv, Normal	Inlet Temp.	1	3312 All Nodes	0.83	0.78	0.9
Exp.13	3320 K	2D Conv, Normal	Inlet Temp.	1	3312 All Nodes	0.78	0.65	0.98
Exp.14	25.1 K	3D Conv, Normal	Inlet Temp.	36	3312 All Nodes	0.87	0.85	0.9
Exp.15	27.1 K	3D Conv, Normal	Inlet Temp. & Power	36	3312 All Nodes	0.8	0.68	0.97
Exp.16	901 K	3D Conv, Normal	Inlet Temp.	36	3312 All Nodes	0.81	0.79	0.83
Exp.17	27.1 K	3D Conv, Normal	Inlet & Outlet Temp.	36	3312 All Nodes	0.8	0.69	0.97
Exp.18	5.4 K	3D Conv, Depthwise	Inlet Temp.	36	3312 All Nodes	0.81	0.69	0.97
Exp.19	7.4 K	3D Conv, Depthwise	Inlet & Outlet Temp.	36	3312 All Nodes	0.81	0.67	0.96

7.6 Integration of Machine Learning Models in a Production System

At UNIBO, a Machine Learning (ML) production framework has been developed for integrating ML models with production systems. This framework consists of three main subsystems: the **monitoring subsystem**, the **ML operation subsystem**, and the **ML anticipation/prediction model**. Figure 7.20 presents the abstraction layers, components,

and software stack of this ML production framework. More detailed information is available in D2.3 and manuscripts[10].

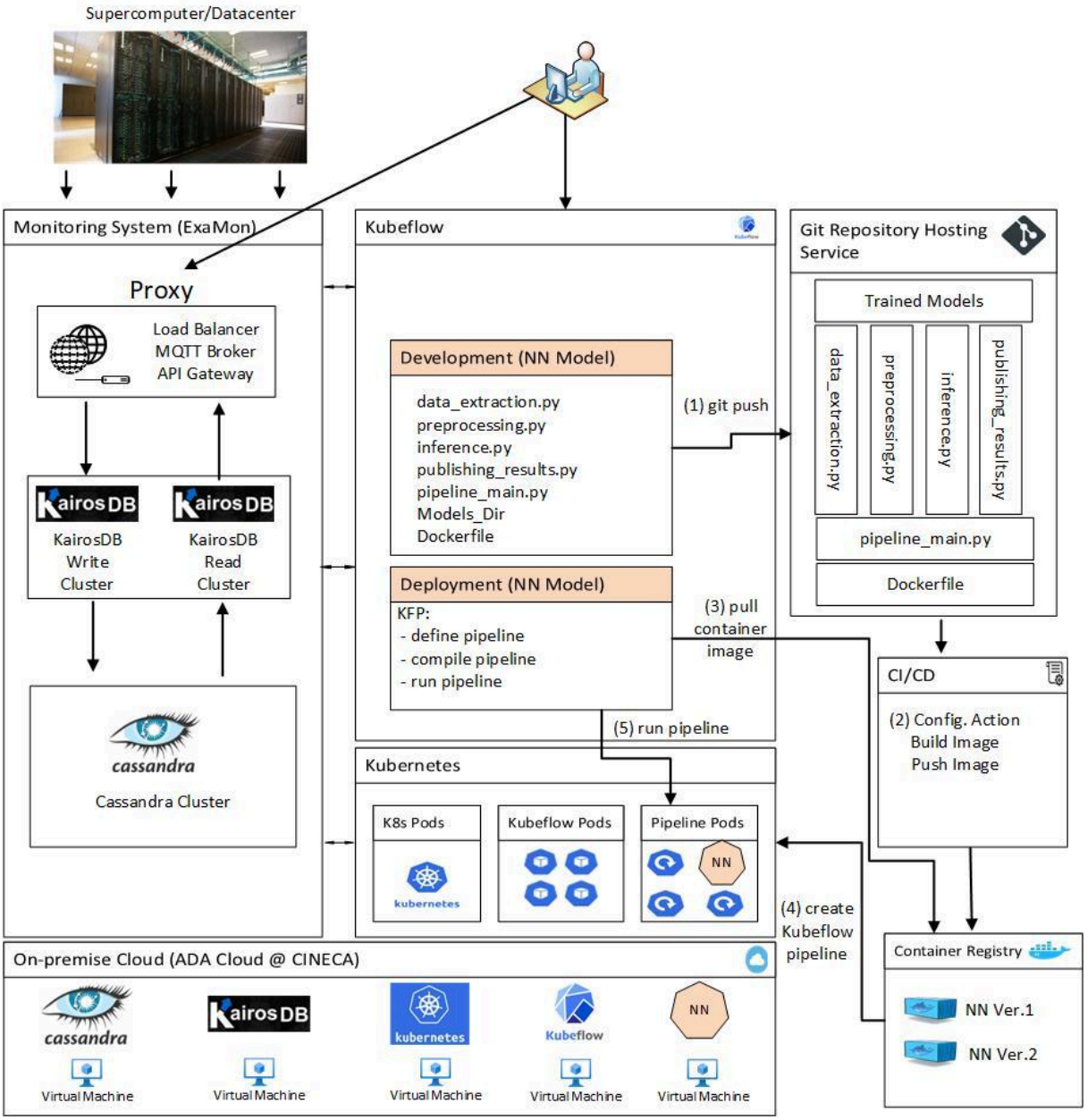


Figure 7.20: HPC monitoring and Machine Learning Operations framework.

Deployment Evaluation

To implement the proposed framework, we employ a cloud system hosted in the CINECA supercomputing facility (on-premise) without creating any overhead on the HPC nodes. This cloud infrastructure is based on the OpenStack version of Wallaby. The nodes of this cloud system are composed of Dual-Socket Dell PowerEdge servers, 2xCPU 8260 Intel CascadeLake processors (24 cores, 2.4GHz), 48 cores per node, hyperthreading x2, 768GB DDR4 RAM, and an internal network of Ethernet 100GbE. The OpenStack virtual machine

[10] Molan, Martin, Mohsen Seyedkazemi Ardebili, Junaid Ahmed Khan, Francesco Beneventi, Daniele Cesarini, Andrea Borghesi, and Andrea Bartolini. "Graafe: Graph Anomaly Anticipation Framework for Exascale Hpc Systems." Available at SSRN 4713330.

executes the ExaMon production, which we extended with additional ones for the Kubeflow and Kubernetes pods needed by the MLOps. The computational resources available for the ExaMon monitoring systems are 300GB of RAM and 40 vCores. We also collect standard Kubernetes metrics. For implementing the MLOps framework, we used Kubernetes version 1.24 in our framework for automated deployment, scaling, and management of containerized applications. Our Kubernetes cluster has 48 vCPUs and 360 GB of RAM available. For Kubeflow, we used the canonical Charmed Kubeflow version 1.6.

We analyzed three types of neural networks, each requiring varying amounts of input data or data points. The data points must be extracted from the database. As will be indicated in Table 7.3, the data extraction phase is the most time-consuming part of the pipeline. Following this, the pipeline includes steps like data preprocessing, inference, and reporting results back to the monitoring system. These steps may vary in latency or computation overhead depending on the size of the data points and the NN model. These three profiles represent different classes of the Neural Network (NN) model. They require less than 1K, approximately 100K, and around 5M data points. For these three sizes, we evaluated the inference pipeline's computing time, network, memory, and computing cost.

We collected several metrics to evaluate our pipeline, including data extraction latency, preprocessing time, inference computing time, and publishing results latency. These metrics are measured in seconds and presented in Table 7.3. We also monitored resource usage, including CPU and memory usage and the number of pods used. The corresponding metrics are summarized in Table 7.4, which shows resource usage for the baseline setup and the three different types of NN models (in view of input data size); the results are grouped into five sub-tables, reporting the resource usage for the monitoring system/ODA (”ExaMon” sub-table), Kubernetes management, Kubeflow management, user workload not including the NN inference (”User Namespace” sub-table), and the workload due to the NN models pipeline (”ML Production pipeline” sub-table).

Processing time and Computational Resources Overhead

Table 7.3 shows the latency for different pipeline parts. The last column reports the inference rate, measured as the number of inferences per hour each NN model type achieves. In pipelines, the inference rate depends on the processing time and latency of the pipeline. Data extraction is the most time-consuming step in the pipeline. Pre-processing only takes up 1% of the data extraction latency, and inference time is less than 1%. As evident in Table 7.3, when scaling the pipeline from the limited number of data points (e.g., from one rack to all the racks of the Marconi100 supercomputers), we notice that the data extraction time scales sublinearly - while increasing the data request of 50x the query time to the ExaMon monitoring system increases only by ~ 5x. After extracting the data, the data preprocessing step requires the second most computing time, while the inference and result publishing steps take negligible time. This result indicates that the proposed framework can scale to exascale system requirements. Moreover, The pipeline being the bottleneckthe data extraction of more complex models can be afforded with the current system at a negligible cost.

To better understand the implication and cost of the proposed MLOps framework in conjunction with ODA, we collected resource usage data for different parts of the monitoring system, Kubernetes, and Kubeflow without running any pipelines to determine the base load

of the framework (as shown in Table 7.4). By looking at the baseline case (Baseline in Table 7.4), we can notice that the ExaMon ODA framework under normal operations (continuous data collection from the different sensors and dashboards) consumes 3 virtual cores (vcores) and 190GB of memory, while the MLOps framework while not processing any data analytics pipeline uses 75 pods (13 used by Kubernetes, 59 Kubeflow, 3 user namespace), 0.66 vcores and almost 7GBs of memory for its micro-services – almost the 22% more vcores and 4% more memory than the pure monitoring framework. Interestingly, when a real-time ML model pipeline is performed (for ~5M input data points in Table 7.4) for all the nodes of the Marconi100 supercomputer, the ExaMon load increases from 3.08 to 3.41. And the MLOps load increases, from 0.66 vcores to 0.96 vcores with a relatively negligible cost for real-time inference (0.03 vcores). As a result, supporting a real-time ML model in production on the Marconi100 supercomputer requires 30% more vcore resources than merely monitoring it. Of this 30% increase, 11% is attributable to the increased load on the monitoring system, while the remainder is associated with the MLOps component. The ML inference pipeline accounts for less than 1% of the entire overhead, making it ready to scale to larger supercomputers, like exascale systems.

Table 7.3: Processing time and latency of different deployment configurations.

MLOps Pipeline Stage Execution Time [s]						
Data Points	Data Extraction [s]	Preprocessing [s]	Inference [s]	Publishing Results [s]	Total [s]	#Inference /Hour
~1K	4.2	0.11	0.013	0.002	4.325	832
~100K	10.33	0.15	0.014	0.002	10.496	343
~5M	50.238	4.6	0.337	0.06	55.235	65

Table 7.4: HPC monitoring and MLOps framework computation resource requirements and ML model pipeline deployment overhead; the 5 main sub-tables indicate the different framework's components.

Data Points	ExaMon				Kubernetes				
	#vcores	Mem [GB]	Net in [kB/s]	Net out [KB/s]	pods	#vcores	Mem [GB]	Net in [kB/s]	Net out [KB/s]
-	3.08	189.5	6670	6739	13	0.31	0.63	1350	864
~1K	3.35	189.5	6680	7334	13	0.31	0.63	1430	870
~100K	3.59	189.5	9588	7732	13	0.31	0.63	1780	880
~5M	3.41	189.5	8686	7975	13	0.31	0.63	1910	880
Data Points	Kubeflow					User Namespace				
	pods	#vcores	Mem [GB]	Net in [kB/s]	Net out [KB/s]	pods	#vcores	Mem [GB]	Net in [kB/s]	Net out [KB/s]
-	59	0.22	5.44	23	28	3	0.13	0.47	7	1
~1K	59	0.22	5.41	24	30	3	0.2	0.5	8	1
~100K	59	0.23	5.41	26	32	3	0.2	0.91	8	1
~5M	59	0.22	5.41	31	32	3	0.4	1.63	21	1

Data Points	ML Production Pipeline									
	pods	#vcores	Mem [GB]	Net in [kB/s]	Net out [KB/s]					
-	-	-	-	-	-					
~1K	1	0.01	0.4	1	1					
~100K	1	0.01	0.44	1	1					
~5M	1	0.03	1.07	14	1					

7.7 Dynamic optimization of the energy-efficiency of HPC applications

The goal of Bull Dynamic Power Optimizer (BDPO) is to optimize the energy-efficiency associated with the executions of HPC applications. To do so, it resorts to fine-grain monitoring of CPU performance counters to identify phases with low computational intensity at runtime. When so, BDPO enforces Dynamic Frequency Voltage Scaling (DVFS) to shift the processor tradeoff between reachable performance and dynamic power consumption toward lowering the latter. The underlying rationale is that decreasing the computational power of the CPUs during low computational phases should only have a limited and negligible impact on the performance of the executed HPC application. However, while operating at lower voltage and frequency, the processor draws less power. As a result, the energy-efficiency associated with the execution of the considered HPC application is increased.

The evaluation strategy for BDPO stems quite directly from the rationale of its approach which was just outlined: executing HPC applications with and without BDPO while monitoring the associated Time-to-Solution (TtS) and Energy-to-Solution (EtS), so as to quantify the improvement of the energy-efficiency induced by BDPO.

Key Performance Indicator

Based on the initial description of the Strategic Objectives (SO) and the rationale of BDPO, it appears evident that SO1.4 (decreased energy-to-solution) is the most relevant objective for evaluating BDPO. However, the Key Performance Indicator (KPI) associated with SO1.4 in Table 3.1, namely the EtS, is not sufficient on its own to evaluate BDPO. It is essential to consider the impact on Time-to-Solution when assessing any decrease in Energy-to-Solution. Therefore, in order to determine the performance of BDPO, we evaluated the reduction in EtS achieved by its actions while also considering an upper bound on the resulting performance degradation. Specifically, we imposed a constraint that the Time-to-Solution should not increase by more than 3% as a result of the actions taken by BDPO.

Experimental platform

The experimental platform used to evaluate BDPO is an Atos on-premise supercomputing partition consisting of 32 BullSequana X2410 compute nodes (details below) with an Infiniband HDR100 interconnection network. The partition is managed by Slurm (version 23.11), and both Bull Energy Optimizer (BEO) and its module implementing TAAPC are installed on the management nodes. BEO features 1Hz out-of-band energy monitoring of the compute nodes, and has the capability to enforce power caps on the latter, also in an out-of-band fashion.

Details of the BullSequana X2410 compute blades:

- 2 sockets with AMD Epyc 7763 (2 x 64 cores) @ 2.45 GHz
- 256 GB DDR4 (16 x 16 GB) @ 3200 MHz
- Direct Liquid Cooling (DLC).

Evaluation protocol

The evaluation protocol for BDPO consists in executing a corpus of HPC applications with different configurations regarding DVFS, while monitoring the TtS and EtS associated with those runs (BEO is used for energy monitoring). The below list described those DVFS configurations:

- **Perf:** it refers to executions with the `acpi-cpufreq performance` governor in charge of managing the frequency of the processors. It acts as a reference (i.e. a baseline) to compare to since it is the default configuration enforced on a vast majority of production HPC systems (including the top supercomputers of the TOP500 rankings) ;
- **Bdpo:** it refers to executions for which BDPO manages the frequency of the processors.

Additionally, the aforementioned set of HPC applications considered in this experimental work is the following: NEMO, NAMD, QuantumEspresso, and GROMACS.
Finally, just as for the evaluation of TAAPC, let's mention the fact that each "experimental point" (i.e. an application executed for a DVFS configuration) was replicated, here 11 times. The cross markers represent the average values of those 11 repetitions.

Results and discussions

Figure 7.21 shows the Energy-to-Solution (EtS) in Joules and the Time-to-Solution (Tts) in seconds for executions of four HPC applications, with BDPO ("bdpo" configuration - purple markers) and without BDPO ("perf" configuration - red markers) being executed in parallel with the applications. The cross markers represent the average values of the associated sets of points. Table 7.5 sums up the results presented by Figure 7.21. As stated above, the "perf" configuration acts as the reference/baseline against which the values associated with the "bdpo" are evaluated to build the "rel" row. For this row, +12.5% in one of the TtS columns would mean that the average TtS for an execution of an application with the "bdpo" configuration is equal to 1.125 the average TtS for an execution of the same application with the "perf" configuration (and, respectively, for -8.75% it would be 0.9125).

Table 7.5: Average Energy-to-Solution (EtS) and Time-to-Solution (TtS) for executions of four HPC applications for the configuration "perf" and "bdpo" regarding the management of the frequencies of the processors of the compute nodes.

	GROMACS		NEMO		QE		NAMD	
	EtS (MJ)	TtS (s)	EtS (MJ)	TtS (s)	EtS (MJ)	TtS (s)	EtS (MJ)	TtS(s)
perf	8.75	607.2	6.60	518.9	21.3	1607	5.39	415.0
bdpo	8.62	597.7	6.57	525.9	21.0	1611	5.23	409.0
rel	-1.5%	-1.6%	-0.45%	+1.4%	-1.4%	+0.2%	-2.8%	-1.4%

Globally, for the 4 applications, using BDPO does not significantly degrade the performance: the increase of the TtS is under the fixed upper bound of 3%.

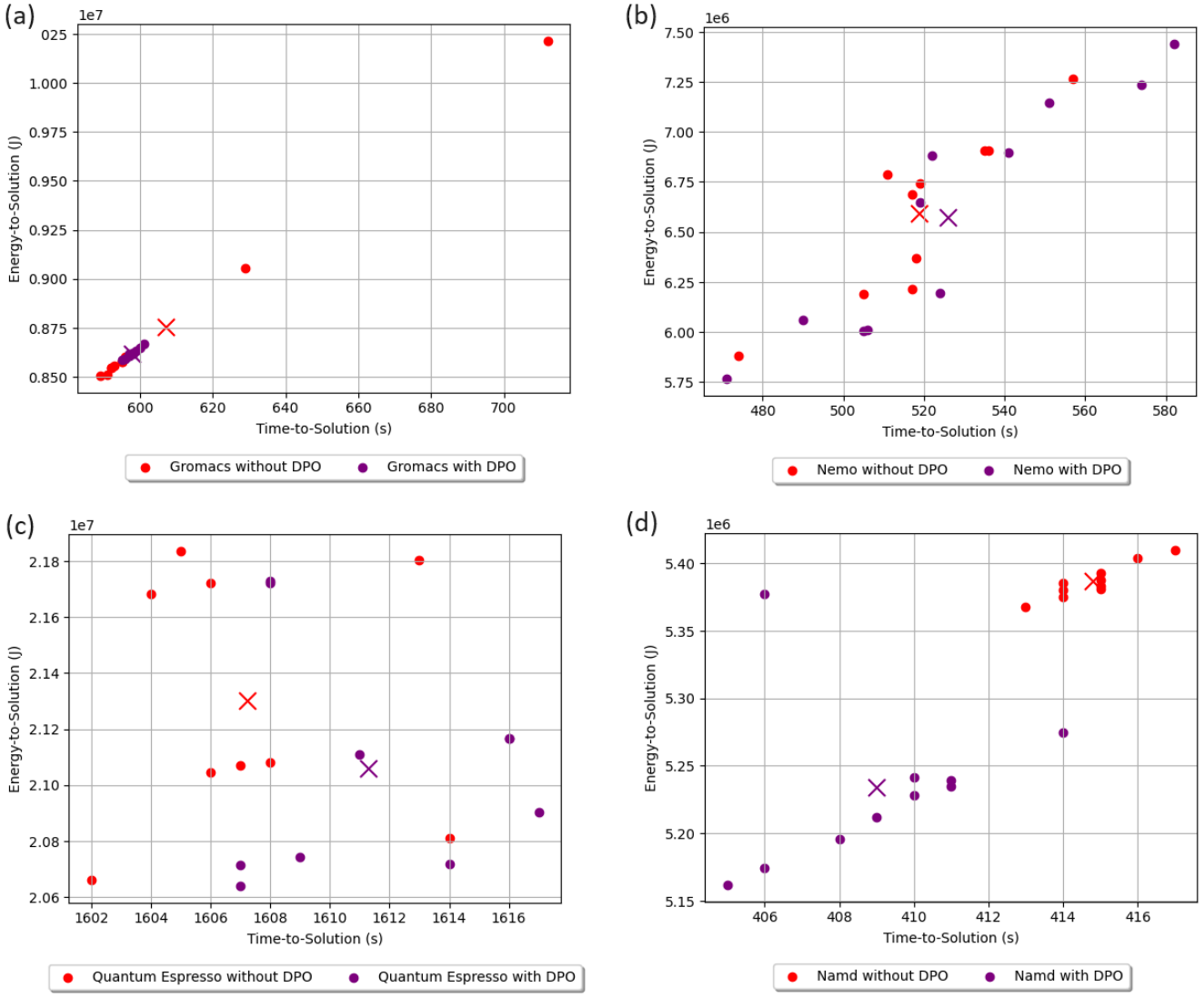


Figure 7.21: Energy-to-Solution (in Joules) and Time-to-Solution (in seconds) with and without DPO for four applications (a) GROMACS (b) NEMO (c) QuantumEspresso (d) NAMD

In detail, GROMACS (due to variability issues on the side of the application, see outliers on the top right corner of Graph (a) of Figure 7.21 and NAMD yields better performance when BDPO is running in parallel with the application. Regarding QuantumEspresso, the 0.2% increase of the TtS is arguably in the “OS jitter” (i.e. the performance variability induced by the execution of the kernel of the operating system in the background), which is tantamount to “no noticeable performance degradation”. Only NEMO performs better without BDPO, which induces an augmentation of 1.4% of the TtS. From performance to energy concerns, let’s lay emphasis on the fact that for the four applications, the EtS is reduced by the action of BDPO, by up to 2.8% for NAMD. Thus, using BDPO tends to improve the energy-efficiency of real-world production-ready HPC applications.

Nonetheless, less than 3% of energy savings, even at no significant cost on the performance side, seems underwhelming. To explain this fact, we note, first, that the architecture of the compute nodes plays a significant role. Indeed, AMD processors exhibit a very limited range of available frequencies. In this case, only 3, namely 1.50 GHz, 2.00 GHz, and 2.45 GHz (with possibility to activate boost frequencies). The large gaps between those frequencies do not make it possible to fine-tune the configuration of BDPO regarding CPU frequencies. And since its action must be as transparent as possible from the performance point of view, achieving high numbers for energy savings without a significant increase of the TtS is not possible. That being said, BDPO should be able to induce better energy savings without additional performance degradation. Should, since, as explained in the **Deliverable D2.3** “Final integration of sophisticated policies in the REGALE prototype”, BDPO is not “complete” yet. Indeed, it is being completely reimplemented to support new processor architectures. To put it roughly, the version of BDPO used for the experiments presented in this document corresponds to a transient state on the path of its refactoring, only packing its core features in their simplest apparel. Once the new major version of BDPO is ready to be released, Eviden intends to repeat an extended version of this evaluation protocol and to disseminate the results in a white paper.

Lastly, one interesting thing to note regarding the experiments with NAMD: enforcing DVFS with BDPO induced, on average, both better performance and lower energy consumption. Indeed, it might seem counterintuitive that scaling down CPU frequency could improve the performance of an application. The underlying explanation of this observation stems from the concept of "thermal headroom". In a few words, when the "perf" configuration is enforced, as soon as it is thermally possible, boost frequencies are enforced. As a result, even in memory-bound phases, the processors are heating up while the boost frequencies do not induce any performance increase. Consequently, during compute-bound phases when boost frequencies would improve performance, there is not enough thermal headroom to enforce the boost frequencies on long time periods. On the contrary, when the "bdpo" configuration is enforced, the frequency of the processors is downscaled during the memory-bound phases, which allow the CPU to cool down. Consequently, the boost frequencies can be enforced for longer periods during the compute-bound phases, which yields better performance for the execution of NAMD.

7.8 Elasticity for Big Data applications

The particular sophistication is structured around the functionalities of the BeBiDa software enabling Big Data applications to be executed elastically upon rigid HPC environments. Typical evolving applications (where malleability is controlled by the application itself) are not supported by default on the traditional HPC resource managers (SLURM, OAR, etc). Hence mechanisms like BeBiDa have been proposed to enable the execution of this type of dynamic applications with no interference to the HPC resource manager, by exploiting the cluster's unutilized resources to submit parts of the Big Data applications dynamically as low-priority preemptable jobs while removing the ones whose resources may be needed by higher-priority typical HPC jobs. In particular, we have focused on Big Data Spark streaming applications. Spark applications are basically controlled by the Spark driver and multiple workers that work in parallel, which are spawned as the applications' needs appear. The Spark application is executed dynamically using the rigid job-based execution mode of HPC resource manager to launch each Spark worker as a different low-priority job. The fault-tolerant and dynamic nature of Spark makes sure that the Big Data application will eventually terminate and BeBiDa handles the usage of the HPC cluster resources elastically. BeBiDa performs the growing and shrinking whenever needed considering both the needs of the application and that of the HPC system but it has the limitation that the Big Data job may be continuously interupted by the higher priority HPC jobs having an important impact on the turnaround time. The sophistication of BeBiDa in the REGALE project has as goal the minimization of the turnaround time of Big Data applications through two new techniques: i) deadline-aware and ii) time-critical.

Key Performance Indicator

The main KPI to track for the particular sophistication of Big Data applications' elasticity is the optimization in turnaround time for Big Data applications while executing them in parallel with traditional rigid HPC applications. Some other indirect KPIs which are related to this sophistication are those of scalability of executions of Big Data applications on HPC resources along with the waiting time and starvation related to both Big Data and HPC applications running on HPC clusters.

Experimental Platform

For this we have been using three different experimental platforms:

1) Emulation environment deploying 3 lightweight virtual machines on one single server representing: i) Kubernetes cluster with 1 master and 1 worker nodes, ii) an HPC cluster with OAR or SLURM with 1 master and 2 compute nodes.
2) A group of servers on an ICCS-hosted platform deploying i) 4 virtual machines for a Kubernetes cluster and ii) 4 bare-metal nodes deploying SLURM with 1 master and 4 compute nodes
3) A group of servers on Grid5000 deploying i) at least 4 VMs for a Kubernetes cluster and ii) 16 or more bare-metal nodes deploying OAR or SLURM for the HPC offloading

The initial developments and preparation of experimentation procedure has been done in all the three platforms but in the end we present here only the results related to the Grid5000 experimentation.

Evaluation Procedure

The goal is to evaluate the new techniques and heuristics that we have implemented to improve the quality of service for the application that are running with BeBiDa. These applications are running using Idle HPC resources but be preempted at any time by a starting HPC job, which implies an undefined execution time if the HPC cluster is loaded.

To cope with this issue, we developed some mechanisms that allow the provision of dedicated resources to the Bebida application and thus improve its quality of service. The mechanisms are described in detail in Deliverable D2.3.

The first mechanism called deadline-aware or *Punch* is to watch the Kubernetes queue to detect pending applications. When detected, a job is created in the HPC queue with a special attribute that tells the HPC prologue to leave the Kubernetes daemon running, thus, during this HPC job the resources are dedicated to the Bebida applications. To determine the size of the Punch job in number of resources and time, the Bebida applications are annotated with resource requirements. Another requirement is the optional deadline that a user can provide. In this case, the Punch job is delayed to finish just before the deadline in order to avoid unnecessary disruption of the HPC workload.

The second mechanism is called time-critical or *Refill*. It creates a fluid dynamic partitioning of the resource, using resource quota in OAR. The HPC scheduler is configured to always keep a defined amount of resources free of HPC jobs, which make them available for the Bebida applications. These resources are considered fluid because this is not a static partition of the cluster that is reserved, but an amount of resource which is not pinned to a particular set of resources. This is a dynamic partition because the amount of reserved resources can change over time. For now, we use annotations on the Bebida application to trigger the increase of reserved resources and decrease it when the application ends. A more interesting policy would be to use historical data to increase the response time with prescriptive decisions instead of reactive ones, but this is out of the scope of this experiment.

The experiment begins by submitting a specific workload of HPC jobs such as Light-ESP[11] which is an adapted version of ESP benchmark[12] which consists of filling up the HPC cluster with a particular number of jobs of different sizes being sent with a certain arrival order and rate. These jobs will keep an important system utilization but there will remain unutilized resources. Hence, the goal is to make use of the unutilized resources for the Big Data Spark job to be executed. So in parallel we submit a specific BigData Spark job related to the Regale pilots, which is being submitted through the workflow engine Ryax to simulate the real execution of the pilots. Once this is submitted the system goes through the integration of BeBiDa with Slurm and OAR and the Spark job makes use of the unutilized resources of the HPC cluster. In this context, we compare the new sophistication techniques (deadline-aware and time-critical) with the default BeBiDa mode and we study the impact the different techniques have on the turnaround time of the Big Data Spark job.

[11] Yiannis Georgiou and Matthieu Hautreux. “Evaluating scalability and efficiency of the Resource and Job Management System on large HPC Clusters”. In: JSSPP. 2012 (cit. on pp. 13, 39, 65, 68).
[12] Adrian T. Wong, Leonid Oliker, William T. C. Kramer, Teresa L. Kaltz, David H. Bailey: ESP: A System Utilization Benchmark. SC 2000: 15

The methodology makes use of nixos-compose to perform reproducible experiments.

Heuristics

This experiment is designed to evaluate the impact of different variants of the aforementioned mechanisms. We defined the following heuristics:

- **NoHPC:** In order to have a control experiment, run the Bebida app without HPC workload
- **None:** Raw Bebida implementation without optimization
- **Punch:** Create jobs at submission time with arbitrary resource requests
- **Refill:** Use resource quota to dedicate a dynamic set of resources to time critical Bebida applications

The Deadline goal is to run before a given deadline. Because it does not follow the same objective it is not to be compared to the others.

Workloads

For this experiment, we run a Spark example application (SparkPi) 4 time in a row using the same heuristic, with 5 sec between each run.

Meanwhile, we emulate an HPC workload generated by LightESP. It starts 10sec before the first Spark application, so the cluster is already loaded with HPC job when the first application starts, and still submits jobs until the end of the last run.

Platform

The platform used for this experiment is Grid5000. We have used 16 nodes on the cluster named “Gros” in the Nancy site. Details of the cluster are available here[13]

The deployment of the system is done in a reproducible manner using NixOS-Compose. It allows use to easily deploy a reproducible software environment defined in a declarative way called a composition. The composition used for this experiment is available here. The environment created contains includes an HPC scheduler (OAR is used for this experiment but Slurm is also available), a lightweight Kubernetes distribution called k3s, and the Bebida optimization service which implements the heuristics.

Results

The following figures show the results retrieved from the experiments. We have reproduced the same experiments 20 times for each heuristic so besides mean times we also show the standard deviation. Figure 7.22 provides the turnaround times of Big Data Spark application being executed with BeBiDa on HPC resources using different heuristics. In particular it shows that the application running alone (nohpc) has a stable execution time. Using Bebida without optimization (none) the Bebida applications runs longer with large uncertainty on the execution time which is expected because it depends on small time frames that sometimes match task execution and sometimes dont. Hence the fault-tolerance of Spark is activated more times than the other cases which provokes more delays in executions.

[13] https://www.grid5000.fr/w/Nancy:Hardware#gros

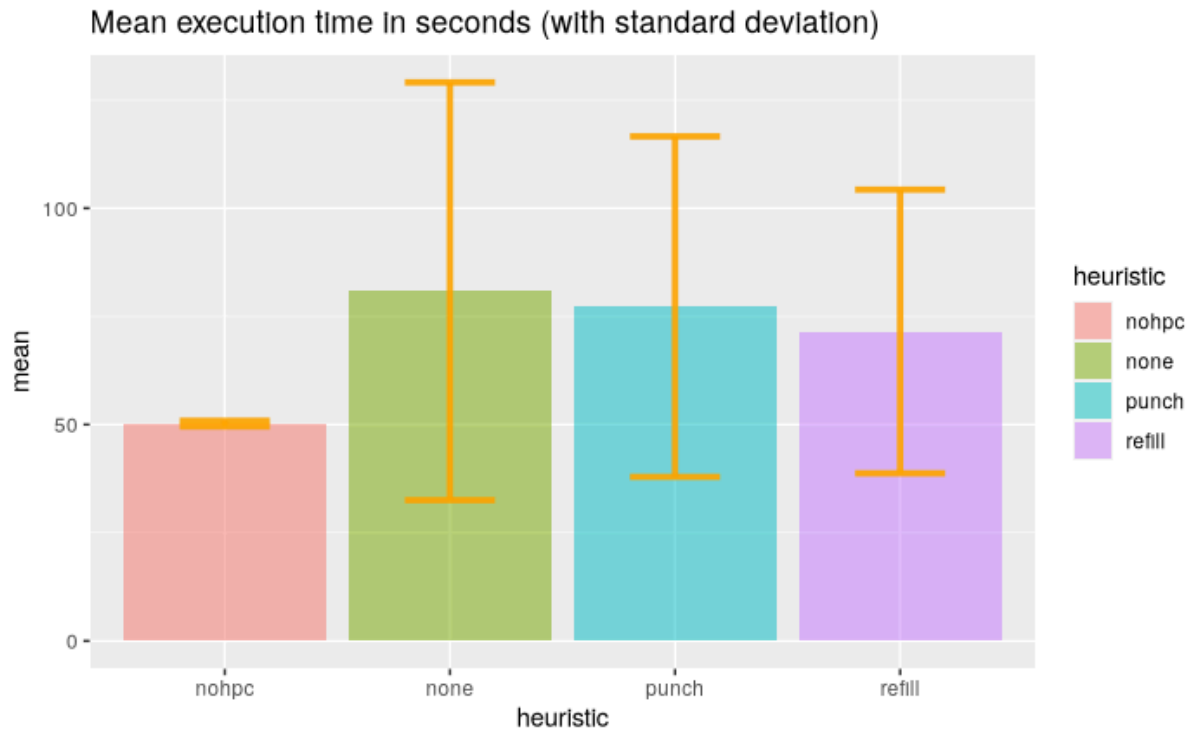


Figure 7.22: Turnaround times of Big Data Spark applications on HPC resources comparing different BeBiDa heuristics

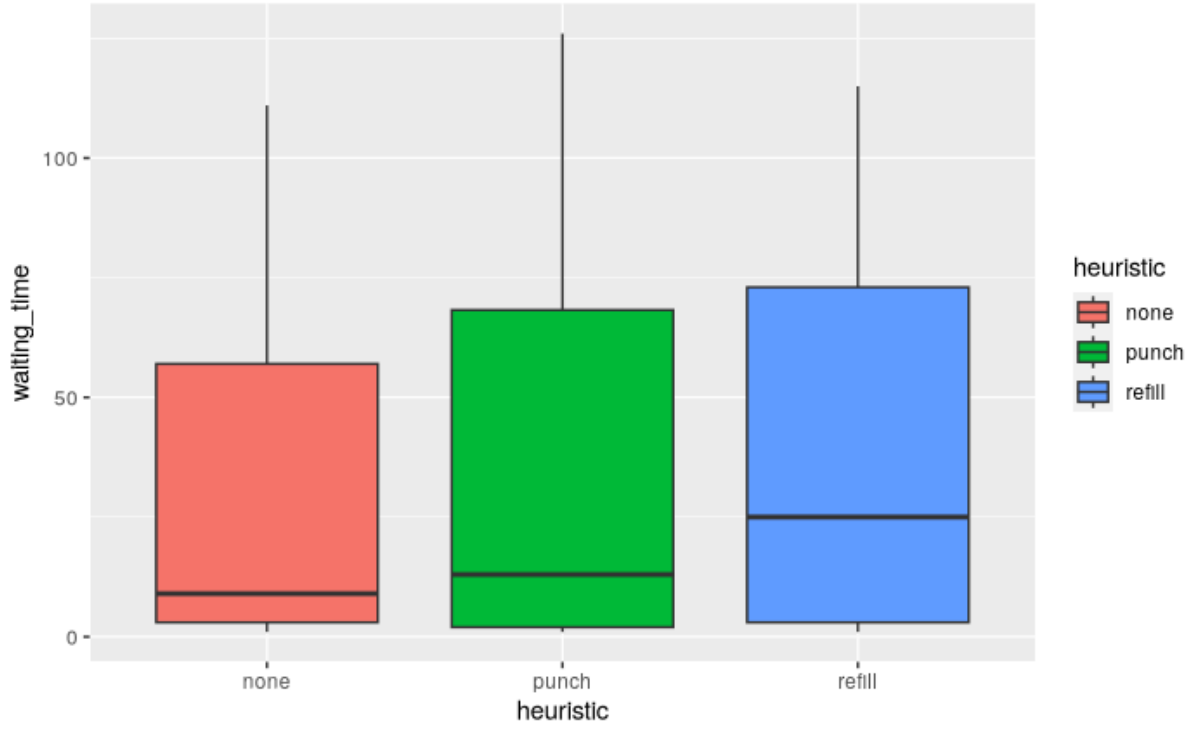


Figure 7.23: Waiting times of HPC jobs when applying different BeBiDa heuristics for execution of Big Data Spark applications

Regarding the heuristics proposed in this work, Figure 7.22 shows that for the Punch method (deadline-aware variation), it reduces the mean execution time by 4,4% when compared to the None heuristic but it also significantly decreases the variability (-18%). The Refill method (time-critical) improves the mean execution by 11,5% and decreases the variability by 32%.

Figure 7.23 shows the comparison of waiting times for different BeBiDa heuristics. This figure shows the impact of BeBiDa heuristics, for execution of Big Data Spark applications, upon the waiting times or starvation of HPC applications. We can clearly observe the impact on the HPC jobs'. The waiting time for None case is explained by the typical BeBiDa overhead in the waiting time of HPC jobs. In the case of Punch jobs this waiting time becomes larger than None since it injects jobs in the workload to reserve resources for Big Data applications. Finally in the case of *Refill* it reduces the amount of available resources for typical HPC executions, which increases even more the waiting time. Of course, this is expected because in order to minimize the turnaround time of Big Data jobs it is normal that the waiting time of HPC jobs would increase.

Finally, another important aspect is the scalability of the techniques. BeBiDa system manager just makes use of specific parameters and mechanisms (prolog/epilog scripts, reservations, etc) of traditional HPC resource managers Slurm and OAR to give the control of their HPC resources to the external resource manager Kubernetes which is adapted to run Big Data applications with higher elasticity. Therefore the scalability of BeBiDa and the way elasticity will be done scalably for Big Data applications is actually directly related to the scalability of the underlying tools Slurm, OAR and Kubernetes. All three of them have been proven to be scalable in different contexts, hence we do not expect particular issues related to that. This said, Kubernetes has not been designed with HPC scalability in mind, but the community is pushing towards more scalable converged computing based on Kubernetes[14], hence our techniques are going to leverage the outcomes of this movement.

Conclusions and Future Work

This section provided the experimentation of the Big Data applications elasticity and in particular the evaluation of the sophistications provided on BeBiDa system manager to improve the quality of service for the Big Data applications being executed with BeBiDa. We have provided a reproducible experimentation methodology making use of NixOS-compose software and different platforms such as Grid5000, upon which we evaluated the new techniques and integrations. Our results are promising and showed a clear improvement in the elasticity and quality of service for Big Data applications when applying the particular techniques.

The execution of Big Data or AI applications upon HPC clusters is becoming an interesting requirement for different use cases. Hence our goal is to continue working on the improvement of BeBiDa. We would like to extend the time-critical (Refill) technique with mechanisms that take into account historical data and eventually making use of ML to further improve the way that we allocate resources for Big Data applications. Furthermore, we would like to extend NixOS-compose mechanisms to be able to deploy experiments upon typical Cloud infrastructures in order to be able to make our experiments upon Cloud resources. Finally we are going to adapt BeBiDa to enable a tighter integration with Kubernetes, Slurm and OAR in order to minimize the waiting times and the impact of collocation upon both HPC and Big Data jobs.

[14] Daniel J. Milroy et al. One Step Closer to Converged Computing: Achieving Scalability with Cloud-Native HPC. CANOPIE-HPC@SC 2022: 57-70

8. Evaluation of qualitative objectives

This deliverable presented an evaluation for the tasks within REGALE that aim to reach the project's quantitative objectives, in particular SO1 and partially *scalability* from SO2. For the sake of completeness, we briefly comment below how we assess the status of the qualitative objectives and refer to the deliverables where more information is provided.

SO2: Platform independence. Platform independence is validated through the capability of instantiation of alternative integration scenarios involving different hardware architectures and components. The REGALE architecture and API definition (Deliverable 1.3), the REGALE integration scenarios and the REGALE library (Deliverable D3.3) have considered platform independence and support this objective.

SO2: Extensibility. Extensibility is sought by the ease of incorporation of new features and alternative modules. This is sought by the REGALE API definition and the REGALE library. More information is provided in Deliverable D3.3.

SO3: Automatic allocation of resources. Atomic allocation of resources refers to the integration of the five REGALE pilots with the relevant workflow engines. The results of this integration are reported in D4.3 and in Section 5.

SO3: Programmability. This has been assessed by the application developers and pilot users of the Consortium by comparing the features of their applications before and after the optimizations within REGALE. Details are provided in D4.3.

SO3: Flexibility. This objective is validated by the ability to execute under lightweight virtualization within the REGALE-enabled system. This has been accomplished by the integration of pilots 3 and 4 with the RYAX workflow engine (Deliverable D4.3).

9. Conclusions

Deliverable D1.4 presented evaluation results from a wide campaign of experimentation across all the activities of the REGALE project, i.e. prototyping, integration of pilots with the workflow engines and sophistication. The evaluation process was designed with a primary goal to assess the status of the project objectives at the end of the project. Based on the results presented, we may note that:

a) REGALE met its ambitious goals to a large extent;

b) Further experimentation is required where simulation was employed, to test and validate the results of REGALE on real systems and at larger scales.

REGALE partners remain heavily involved in all activities of the project (tools, prototypes, pilots and sophistication) and their continuation and exploitation plans ensure concrete paths towards incorporation of the project results in production systems.

Appendix A: Co-scheduling simulator

Co-scheduling HPC Simulator

The simulator's end goal is to provide a tool (API and dashboard) for rapid development and experimentation on scheduling and co-scheduling algorithms for HPC systems.

How does it work?

The simulator is divided into four main components:

1. The *Generator* creates a set of jobs that is used as input for the simulation.
2. The *Cluster* handles the logic of executing the jobs and moving the simulation step by step to the next state.
3. The *Scheduler* provides the policy on how to submit the jobs for execution.
4. The *Logger* records cluster and job level events for a run and provides end-points for plotting.

The cluster, scheduler and logger are the computational backbone of the simulation.

Generator

The Generator component creates a set of jobs as input for a simulation run, given a set of real runs with real workloads on a HPC cluster. It wraps the information about each of their execution time and compiles some additional hints. These are a unique numeric identifier (job id), a job name, the number of processes asked, the current speedup of the job and current binded cores.

In order to produce a set of jobs, there are three different techniques provided. The first is the random selection of workloads from a set of real runs given as an input their number of populace. The second, is to create a set of jobs given the occurrence of each individual workload. The third is given a list of workloads' names to create a set of jobs based on it.

Each of these techniques gives a certain degree of freedom to the user. The first technique doesn't provide any degree of choice other than the workloads the set of jobs is based on. The second technique gives the freedom to choose which workloads will appear as a job. The third gives the user the control to choose the workloads and their position/arrival in the set of jobs.

Cluster

The Cluster component is considered the critical linking point of the simulation. It contains the architectural information about the simulated HPC cluster. It handles the resource consumption of the jobs and the general steps of the simulation until it finishes.

At each step the cluster checks if there are any jobs left in the waiting queue. If there are, then it tests whether the scheduler can submit a new batch of jobs for execution. If there are some jobs left in the waiting queue but the scheduling algorithm cannot submit any of them,

then the cluster starts “executing” the jobs inside the execution list. By “executing”, it means finding the job with the smallest remaining execution time, removing this time from the rest of the executing jobs and freeing the resources (binded cores) of the jobs that have finished execution. When resources are freed the cluster calls the scheduler and retries to submit any jobs left in the waiting queue. Likewise, if there aren’t any job in the waiting queue but there are jobs left inside the execution list then the cluster moves to executing them.

The stop condition of the simulation is defined outside the cluster so as the user has complete control of how the simulation is expected to run.

Scheduler

Each scheduler follows a policy on how to submit the jobs in the waiting queue to the execution list of a cluster for execution.

A co-scheduling algorithm needs to concern itself on how to make pairs of jobs and where to find these jobs to make pairs. The only two places available to search for jobs are the waiting queue and the execution list of the cluster. The logic pathways on how to make pairs from both the waiting queue and the execution list are similar. When a co-scheduler is at the waiting queue it queries if there is any job that it must prioritize to submit in order to elevate the performance of the system and what is the best candidate-job to pair it with. When it is at the execution list it queries if there is any executing block of jobs with unbinded cores that it must engage first to improve the overall performance. It also searches for the best candidate in the waiting queue in order to fill the gap of the unbinded cores. These four decision points are approached as heuristics functions. The priority of where to search for the jobs is changeable.

In our experiments we implemented two co-scheduling algorithms. The first one is the Random Co-Scheduler which creates random pairs completely ignoring the heuristics functions. The second one is an implementation of the heuristics co-scheduling architecture. It uses the heatmap of real workloads and other job characteristics, as well as information about the system in order to construct the next pair of jobs for execution.