H2020-JTI-EuroHPC-2019-1

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

Grant Agreement Number: 956560

D1.2

REGALE Intermediate Architecture

Final

Version: 2.0

Author(s): Eishi Arima (TUM), Mohsen Seyedkazemi Ardebili (UNIBO), Georgios Goumas (ICCS), Varvara Asouti (NTUA), Ioannis Kalogeris (NTUA), Ioannis Ledakis (UBI), Ioannis Plakas (UBI), Antonis Koukourikos (SCIO), Bruno Raffin (UGA), Alejandro Ribes (EDF)

Contributor(s): Listed in Acknowledgement section

Date: 05.05.2023

Project and Deliverable Information Sheet

<table>
<tr><td rowspan="7">REGALE Project</td><td colspan="2">Project Ref. №: 956560</td></tr>
<tr><td colspan="2">Project Title: REGALE</td></tr>
<tr><td colspan="2">Project Web Site: https://regale-project.eu</td></tr>
<tr><td colspan="2">Deliverable ID: D1.2</td></tr>
<tr><td colspan="2">Deliverable Nature: Report</td></tr>
<tr><td rowspan="2">Dissemination Level: PU *</td><td>Contractual Date of Delivery:
30 / 09 / 2022</td></tr>
<tr><td>Actual Date of Delivery:
05 / 05 / 2023</td></tr>
<tr><td colspan="2">EC Project Officer: Evangelos Floros</td></tr>
</table>

* - 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

<table>
<tr><td rowspan="6">Document</td><td colspan="2">Title: REGALE Requirements, Initial Architecture, and Evaluation Plan</td></tr>
<tr><td colspan="2">ID: D1.2</td></tr>
<tr><td>Version: 2.0</td><td>Status: Final</td></tr>
<tr><td colspan="2">Available at: https://regale-project.eu</td></tr>
<tr><td colspan="2">Software Tool: Google Docs</td></tr>
<tr><td colspan="2">File(s): REGALE_D1.2_Architecture_ver2.0.pdf</td></tr>
<tr><td rowspan="4">Authorship</td><td>Written by:</td><td>Eishi Arima (TUM), Mohsen Seyedkazemi Ardebili (UNIBO), Georgios Goumas (ICCS), Varvara Asouti (NTUA), Ioannis Kalogeris (NTUA), Ioannis Ledakis (UBI), Ioannis Plakas (UBI), Antonis Koukourikos (SCIO), Bruno Raffin (UGA), Alejandro Ribes (EDF)</td></tr>
<tr><td>Contributors:</td><td>Listed in Acknowledgement section</td></tr>
<tr><td>Reviewed by:</td><td>Georgios Goumas (ICCS), Andrea Bartolini (UNIBO), Martin Schulz (TUM)</td></tr>
<tr><td>Approved by:</td><td>Georgios Goumas (ICCS)</td></tr>
</table>

Document Status Sheet

Version	Date	Status	Comments
0.1	21.09.2022	Draft	Initial version
0.2	28.09.2022	Draft	Internal review completed
0.3	04.10.2022	Semi Final	Major updates
1.0	06.10.2022	Final	Final fixes
1.1	1/3/2023	Deliverable reopened	Structure of evaluation plan
1.2	15/3	Draft	Initial evaluation plan
1.3	31/3/2023	Draft	Evaluation plan detailed
1.4	21/4/2024	Draft	Final evaluation plan
1.9	28/4/2023	Semi-Final	Minor edits
2.0	05/05/2023	Final	Final edits

Document Keywords

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

Executive Summary

This deliverable document reports the current status of WP1 (work package on requirements, architecture and evaluation) in the REGALE project, i.e., the requirement specifications (Task 1.1), the intermediate software architecture (Task 1.3). The ultimate goal of REGALE is to pave the way of next generation HPC applications to exascale systems, and to accomplish this we define an open architecture (WP1), build a prototype system (WP3) and incorporate in this system appropriate sophistication (WP2) in order to equip supercomputing systems with the mechanisms and policies for effective resource utilization and execution of complex applications (WP4). We are conducting these studies in a cooperative manner, i.e., the architecture and the prototype are co-designed/conceptualized considering both state-of-the-art and next generation HPC applications, maximizing in this way its applicability. As a first step, this document describes the architecture requirements for a variety of PowerStack use cases (Section 4) and the intermediate status of the software architecture for PowerStack path (Section 5), which is used as a blueprint for the other work packages, e.g., the software prototyping in WP3 (Section 6). Then, this document concludes with next steps towards updating the open architecture based on the feedback from the other work packages as well as the integration of the different paths in WP1. Finally, the deliverable also included an evaluation plan with a focus on the project's quantifiable Strategic Objectives (Section 7).

Acknowledgement

Here, we declare the contributors to this document and how they exactly contributed as well. As for the authorship, here are the exact roles/assignments:

- **Entire document organization/format:** Georgios Goumas (ICCS), Eishi Arima (TUM)
- **Front matter, Section1 to Section3:** Georgios Goumas (ICCS), Eishi Arima (TUM)
- **Section4 (Use Cases and Requirements):** Eishi Arima (TUM), Mohsen Seyedkazemi Ardebili (UNIBO)
- **Section5 (PowerStack Path):** Eishi Arima (TUM)
- **Section6 (Tool Assessment and Summary of Integration):** Eishi Arima (TUM)
- Section 7 (Evaluation Plan): Georgios Goumas (ICCS), Varvara Asouti (NTUA), Ioannis Kalogeris (NTUA), Ioannis Ledakis (UBI), Ioannis Plakas (UBI), Antonis Koukourikos (SCIO), Bruno Raffin (UGA), Alejandro Ribes (EDF)
- **Section8 (Conclusions and Future Directions):** Eishi Arima (TUM), Georgios Goumas (ICCS)
- **Review Committee:** Georgios Goumas (ICCS), Andrea Bartolini (UNIBO), Martin Schulz (TUM)

The following individuals contributed to defining the high-level interfaces by providing the information with respect to the APIs, CLIs, etc. used in their tools (alphabetized by last name):

Mohsen Seyedkazemi Ardebili (UNIBO), Julita Corbalan (BSC), Julien Forot (ATOS), Abdelhafid Mazouz (ATOS), Michael Ott (BADW-LRZ), Millian Poquet (UGA), Federico Tesser (CINECA)

The following individuals joined our regular meetings for WP1/WP3 at least once, in order to design Regale architecture and clarify the requirements (alphabetized by last name):

Mohsen Seyedkazemi Ardebili (UNIBO), Eishi Arima (TUM, Lead), Varvara Asouti (NTUA), Andrea Bartolini (UNIBO), Daniele Cesarini (CINECA), Christos Charisis (SCIO), Julita Corbalán (BSC), Fanny Dufossé (UGA), Pierre-François Dutot (UGA), Kontoleontos Evgenia (ANDRITZ), Julien Forot (ATOS), Yiannis Georgiou (RYAX), Georgios Goumas (ICCS, Lead), Daniele Gregori (E4), Klaus Hopfner (ANDRITZ), Konstantinos Ioakimidis (ANDRITZ), Ioannis Kalogeris (NTUA), Antonis Koukourikos (SCIO), Giannis Ledakis (UBITECH), Matthias Maiterth (TUM), Abdelhafid Mazouz (ATOS), Hafid Mazouz (ATOS), Martin Molan (UNIBO), Panagiotis Mpakos (ICCS), Alessio Netti (BADW-LRZ), Michael Ott (BADW-LRZ), Nikela Papadopoulou (ICCS), Ioannis Plakas (UBITECH), Millian Poquet (UGA), Bruno Raffin (UGA), Alejandro Ribes (EDF), Olivier Richard (UGA), Martin Schulz (TUM), Mathieu Stoffel (ATOS), Nikolaos Triantafyllis (ICCS), Carsten Trinitis (TUM), Xenofon Trompoukis (NTUA), Marianna Tzortzi (ICCS), Pedro Velho(RYAX), Josef Weidendorfer (BADW-LRZ)

Last but not least, we'd like to express our gratitude to the PowerStack initiative community [1], in particular, Siddhartha Jana (Intel) and Torsten Wilde (HPE) for sharing their insights on the power stack use cases as well as for discussing possible collaboration opportunities.

Table of Contents

1. Introduction

An exascale supercomputer will not be “yet another big machine”. With a cost of hundreds of million euros, power consumption in the order of tens of megawatts and a lifetime that reaches a decade at most, judicious management of those resources is of utmost importance. Turning our attention to the critical aspect of power consumption, the current leader in the TOP500 list as of Jun. 2022 [2], has an exascale computational capacity and a power consumption that exceeds 20MW. Even with the highest technological advancements, a post-exascale machine is expected to well exceed the 20-30MW threshold that is the current upper bound of power consumption for exascale computing, and could be even more than 30MW. A machine of this size will not be able to operate at full power consumption, and energy consumption will become a primary concern to keep its environmental footprint and operational costs at acceptable levels without neglecting its ultimate purpose: to equip highly critical applications with the computational capacity to solve extremely resource hungry problems.

Focusing on the application side, achieving scalable performance and high system throughput has always been a cumbersome task. To make things even more challenging, next-generation HPC applications can no longer be considered as computation-/communication-intensive, monolithic blocks with minimal and infrequent I/O requirements. The revolution of Big Data and Machine Learning, the emerging Edge Computing and IoT, with the scale of modern HPC systems and cloud datacentres, are rapidly changing the way we solve scientific problems. Novel computational patterns are rapidly evolving, where the solution of a problem may require a workflow of diverse tasks, performing simulations, data ingestion, data analytics, machine learning, visualization, uncertainty quantification, verification, computational steering and more. Existing solutions may render the execution of such applications in a large-scale supercomputer either impossible, or extremely suboptimal in terms of time to solution and user cost, due to the absence or inefficiencies of appropriate methods to compose, deploy and execute workflows, and/or due to their extreme requirements in I/O resources, which cannot be met by the system capacity without holistic and sophisticated deployments.

The ultimate goal of REGALE is to pave the way of next generation HPC applications to exascale systems. To accomplish this, we define an open architecture, build a prototype system and incorporate in this system appropriate sophistication in order to equip supercomputing systems with the mechanisms and policies for effective resource utilization and execution of complex applications. The REGALE architecture and prototype will be co-designed considering both state-of-the-art and next generation HPC applications, maximizing in this way its applicability.

REGALE takes an approach that considers two interacting paths: The first path is largely motivated by the PowerStack initiative [1] that primarily targets multi-criteria operation of supercomputing services with a strong focus on power and energy efficiency. The second path focuses on the requirements posed by non-conventional, workflow-based applications and their integration with an appropriate workflow engine, with a goal to achieve easy and flexible use of supercomputing resources at large scales.

This deliverable sets the critical stepping stone for the implementation of REGALE: It starts from the project’s strategic objectives (Section 2), our strawman architecture and software

tools (Section 3), and analyzes a set of relevant use cases together with their requirements (Section 4). These are then used to define the REGALE architecture, components, and interfaces (Section 5) instantiated with the use of the various modules brought in REGALE and evolved throughout the project by the partners (Section 6). Finally, Section 7 concludes the intermediate status of this work and introduces several future research directions to the end or even after the end of this work.

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 will consider the huge 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, 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 disappointing performance, inability to scale, misuse of resources and consequent over charges 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 will be able 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 the 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 will support the operation of a supercomputer with energy consumption as a first class citizen. In this case we will 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 will guide 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 will validate the existence of the following key features:

- **Scalability:** The REGALE system should be able to operate in exascale setups and beyond. To assess this objective we will perform experimental results and simulation, and we will also extrapolate our results to larger system scales. Our goal is for our prototype system to have minimal overheads across all scales.
- **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. This will be validated by our integration process where we will provide full integration scenarios with at least two vendor-specific solutions and will provide indicative solutions for all major modules of the HPC ecosystem.

- **Extensibility:** The REGALE system should be extensible to any new feature or component that aligns to its open architecture. This will be validated through our implementation process. We will build the REGALE system with gradual incorporation of features, starting from the critical ones and adding sophistication and complexity within the various versions in the development and integration process.

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 will 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 will equip them with an easy-to-use set of tools to facilitate the development and deployment of their applications to exascale systems.

To assess if this SO is met, we will validate 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. We will compare the process of requesting resources between the current state-of-the-art systems and applications and the REGALE solution.
- **Programmability:** Application developers should find the REGALE architecture and system easily accessible to develop and deploy their code(s). This will be qualitatively assessed by the application developers and pilot users of the consortium by comparing the features of their application before and after the optimizations within REGALE.
- **Flexibility:** Applications should be able to execute under lightweight virtualization within the REGALE- enabled system.

The architecture, integration and evaluation plans that are presented in this deliverable are driven by REGALE's strategic objectives.

3. Intermediate Architecture and Software Tools

In this section, we first introduce the REGALE intermediate architecture and its components/actors. We second summarize the software tools to be used in this project. We finally introduce our implementation paths that integrate the tools.

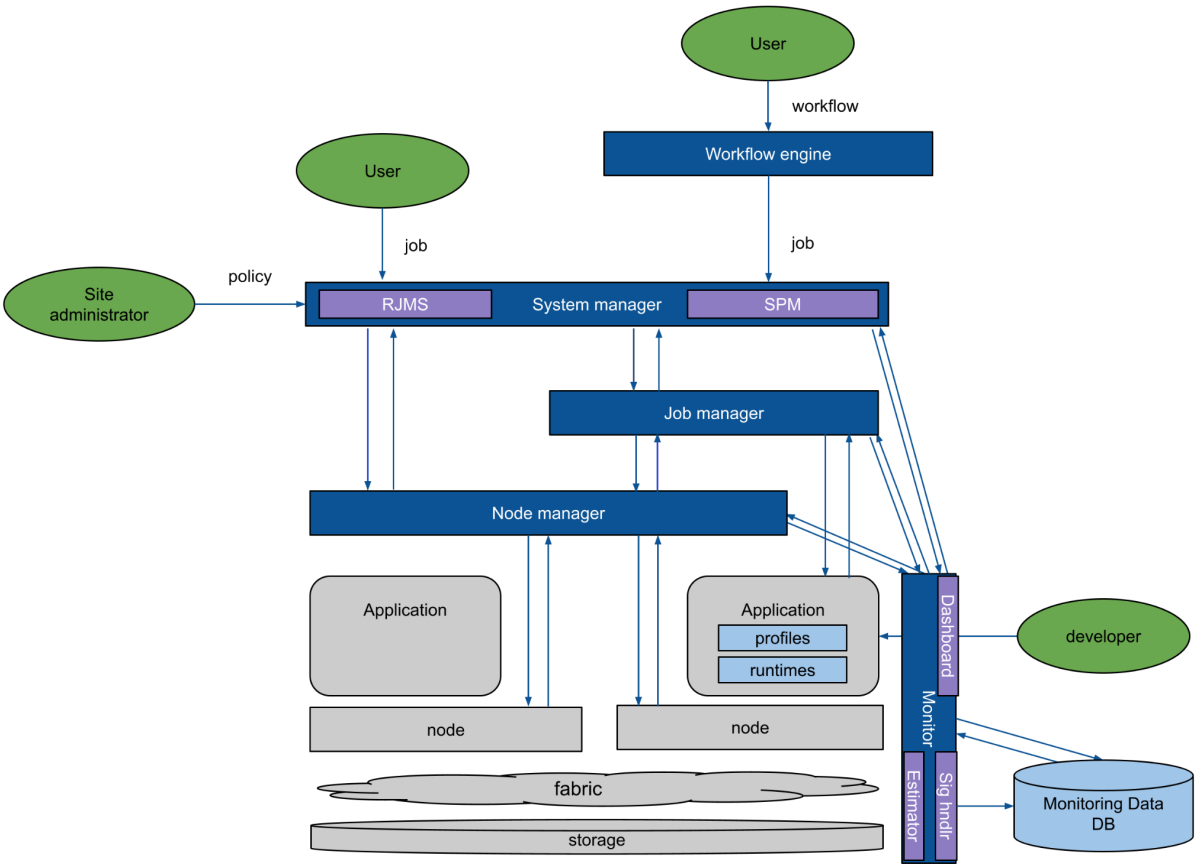

Figure 1: REGALE Intermediate Architecture

Figure 1 illustrates the general intermediate architecture. The descriptions of key actors and software components are as follows.

Human actors:

A. **Site administrator:** Configures the site-level policy appropriately prioritizing between power/energy/performance and quantifies the relevant constraints. The policy can be changed according to the current needs with respect to objectives and/or constraints.
B. **User:** This actor submits a job for execution to the system, requests resources for her job and optionally provides information on the performance behaviour of her application.
C. **Developer:** This actor develops, optimizes and instruments her application with regard to relevant objectives to facilitate further optimization by the system and collection of profiling information.

System modules:

1. **System manager:** The system manager receives as input a set of jobs to be scheduled within the system and indicatively decides upon when to schedule each

job, to which specific compute nodes to map it, and under which power budget or setting. For this, it constantly monitors and records power and energy telemetry data, and controls power budgets/settings and/or user fairness. The system manager applies system-wide optimizations and consists of the following two sub-modules that work cooperatively.

- **Resource and Job Management System (RJMS):** The RJMS manages jobs submitted by users and decides the assignments of node resources to them and their launch timing as well. The decisions are based on the job information given by the users, power/performance features characterized by the Monitor (see next), and the node/job/system power budget information managed by the SPM as well as the scheduling policy given by the site administrator. The decisions are principally made in a static and proactive manner.
- **System Power Manager (SPM):** The SPM manages the power budget allocations across nodes/jobs, including compute nodes, I/O nodes, and others. The SPM provides the functionality to set the power cap to the entire system and also can optimize the power budgeting across nodes depending on the objective/constraints given by the site administrator, while interacting with other modules such as the node manager, monitor, and others.

2. **Job manager:** The job manager performs job-centric optimizations considering the performance behaviour of each application, its fine-grained resource footprint, its phases and any interactions/dependencies dictated by the entire workflow it participates in. It manages the control knobs in all compute nodes participating in the job and optimizes them during runtime to achieve the desired power consumption (at maximum possible performance), efficiency, or other settings. Additionally, it scalably aggregates application profile/telemetry data from each node servicing the given job through the system manager.
3. **Node manager:** The node manager provides access to node-level hardware controls and monitors. Moreover, the node manager implements processor level and node level power management policies, as well as preserving the power integrity, security and safety of the node. For this reason, all the power management requests coming from the software stack are mediated by the node management.
4. **Workflow engine:** The workflow engine analyses the dependencies and resource requirements of each workflow and decides on how to break the workflow into specific jobs that will be fed to the system manager. Modern workflows may be composed of hybrid Big Data, Machine Learning and HPC jobs; hence a key role for the workflow engine is to provide the right interfaces and abstractions in order to enable the expression and deployment of combined Big Data, HPC jobs. The distribution of jobs can vary depending on the objective goals defined by the optimization strategy.
5. **Monitor:** The monitor is responsible for collecting in-band and out-of-band data for performance, resource utilization, status, power and energy. The monitor operates continuously without interfering with execution, with minimal footprint, and collects, aggregates, records, and analyses various metrics, and pushes necessary real-time data to the system manager, the node manager and the job manager. The monitor has the following sub-modules.
    - **Signature Handler[1]:** The signature handler receives the information of job identification from other components and then generates the signature to

[1] Although the signature handler is located inside the monitor module as it accesses profiles or monitored data at runtime, it may be an independent module in an actual implementation. We will continue assessing where to locate this module for the final version of our Regale architecture.

characterize the job. The signature could be calculated with the job information given by the user, the associated job profile of previous runs, or the statistics acquired at runtime.

- **Estimator[2]:** The estimator assesses the job properties (e.g. performance, power/energy consumption, or others) or system status (e.g. anomaly) by using such as the signature generated by the signature handler.
- **Dashboard:** The dashboard provides a set of functionalities that display the node/job status obtained at runtime (or given from a profile) to the developer.

To realize the software architecture, we integrate the following tools. TABLE 1 represents the general classifications of our software tools into the system modules. Note the details of our software tools are described in our previous deliverable (D1.1), except for the following one:

- Execution Profile Compute Module (EPCM) is newly introduced to help our OAR-BEO integration plugin with providing functionalities to estimate the power/performance properties based on the relevant job profile.

TABLE 1: Tool Classifications
X = Supported, / = Under Development, ? = Potential Support

	Monitor				Node Manager	Job Manager	System Manager		Workflow Engine
	DB	Dashboard	Estimator	Sig Hndlr			RJMS	SPM	
SLURM							X		
OAR							X		
DCDB	X	X	/	/					
BEO	X	X			X			X	
BDPO						X			
EAR	X	X	?	?	X	X		X	
Melissa									X
RYAX									X
Examon	X	X	/	/					
COUNTDOWN						X			
PULPcontroller					X				
BeBiDa							X		
EPCM			/					/	

We realize different implementations using the above tools based on the REGALE intermediate architecture, which can be divided into *PowerStack* and *workflow engine paths*, and the latter consists of *Mellissa path* and *RYAX path*. On one hand, the PowerStack path aims to prototype a software stack to enable full-scale production-grade solutions for a variety of power/energy management use cases. On the other hand, the workflow engine paths focus more on the application side, i.e., integrating the workflow management tools

[2] The estimator can also be an independent module or can sit inside of multiple different modules. The location of this module may change in the final version of our Regale architecture.

(Mellissa or RYAX) with our pilot applications as well as other components in our architecture, in order to realize next-generation application management techniques including automatic parameter sensitivity analysis, ML-based simulation surrogate and dynamic concurrency controlling. The PowerStack and workflow engine paths will be first integrated individually because of their different focuses, however we envision combining them in the later stage of the project. In this deliverable document, we mainly focus on the PowerStack path, define use cases or policies with requirements and describe the needed architectural modules. For the workflow engine paths, we introduce their software architecture and the functionalities they support.

4. PowerStack Use Cases and Requirements

In this section, we applied minor updates from the previous deliverable D1.1 based on our continuous discussions in WP1 and ongoing software integrations in WP3. However, note most of the contents and the core concepts are exactly the same as those described in D1.1.

In Task 1.1, we gathered the requirements posed by all key actors in the REGALE architecture, with a particular focus on the PowerStack path. Note the requirements, as well as use cases, components, and interfaces, will be extended to cover all the other paths including Melissa, RYAX, and sophistication paths in the final version of the deliverable. To this end, we conducted our studies in both top-down and bottom-up ways. The top-down approach involved detailed discussions around the possible use cases for the PowerStack path, starting from the most naive one toward much more sophisticated power management schemes, and then we clarified the requirements for each use case. The results are described in this section. The bottom-up approach surveyed the current state of our software tools in terms of what functionalities they support and how they can interact with other tools, in order to gain some insights for the general and open requirements/architecture as well as to confirm the possible use case supports with these tools and how they should be integrated.

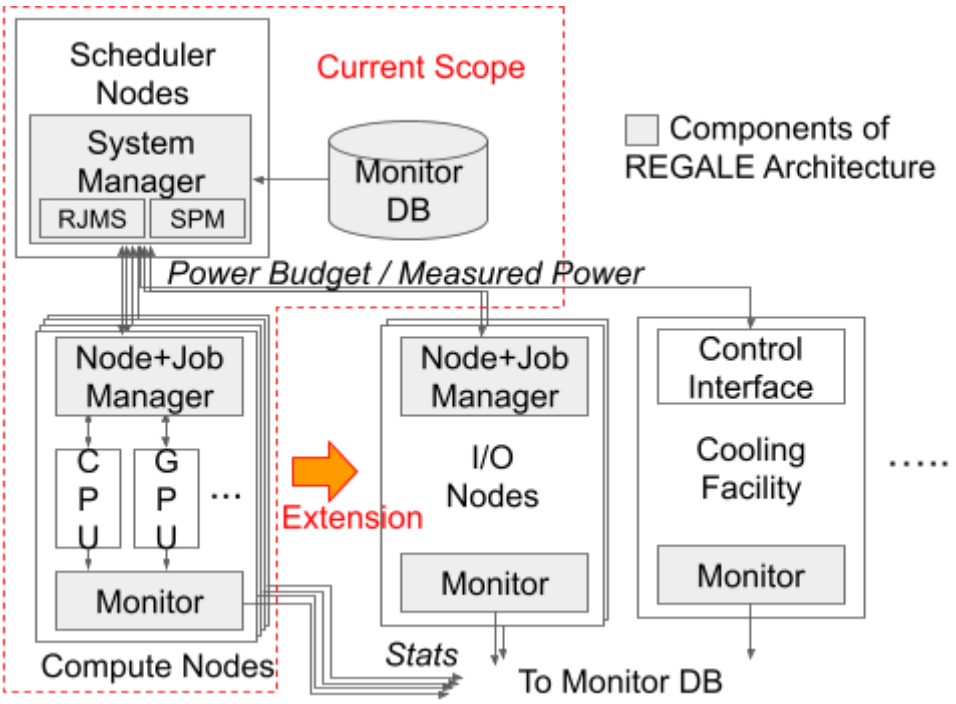

Figure 2: Holistic Power Management and Our Current Scope

4.1 Our Current Scope and Hardware Requirements

Figure 2 illustrates an example of our target HPC systems, assuming holistic power management, and our current scope. The system consists of multiple different high-level hardware components such as compute nodes, I/O nodes and other facilities including the cooling system. In each computer node, there are different components such as CPUs,

GPUs, NICs and DRAM memories. The overall power management is governed by the system power manager daemon launched on the scheduler (or admin) nodes. More specifically, the system manager (SPM) distributes power budgets across nodes or any other target facilities, which could be in a closed or open loop manner. The closed-loop control makes power budget decisions based on the actual power consumptions, while the open-loop option does not utilize them. The node/job managers and the monitor are distributed across the nodes, and they are responsible for the power setups on node components and the measurement within a node. In this period of time, we focus on the power management only on *compute nodes* as they are generally the major power consumers in HPC systems. In other words, we fix the power budget (or limit) setups on the other kinds of nodes or facilities at their maximum. In the future work, we intend to include the other components and scale down/up their power budgets depending on their utilizations.

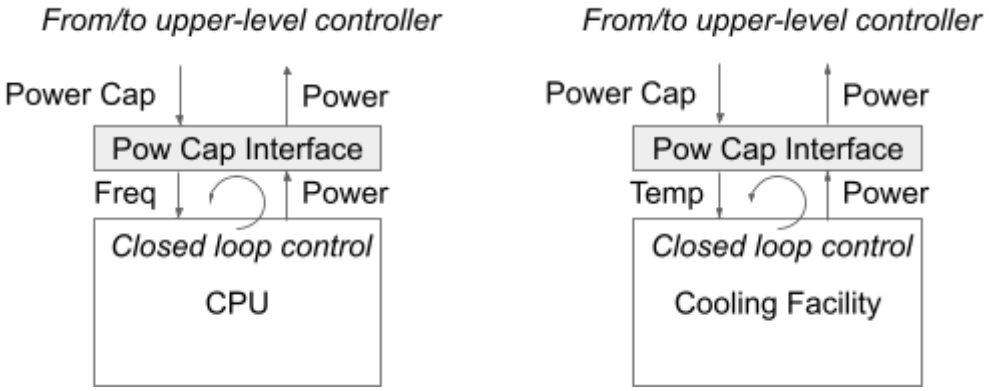

Figure 3: Power Capping Interface Requirements

In order to handle global power budgets across different kinds of components/facilities, we need to define and unified currency for the power exchanges. More specifically, different components have different knobs to trade-off performance against power (e.g., clock frequency for CPUs or temperature setup for cooling facilities), here we call them as *power management knobs*, and these knobs can change or can be extended/removed for future products. Therefore, directly controlling them from the highest-level component, typically the system manager, would not make sense, and the hardware specific aspects should be hidden as much as possible.

For this reason, we set the following requirements for hardware components to include them for the power budgeting: (1) they (or other low-level software layers such as operating systems) must provide a software interface to set power cap (or limit) to them, which must be controllable from the PowerStack software; (2) they must periodically monitor the actual power consumption and adaptively control the hardware knob that can scale down/up the power consumption as a function of activity or throughput, in order to enforce any given power cap; and (3) the interface must also provide the actual power consumption to the overlying software stack. The third point is not needed for open-loop power management use cases, but is required for closed-loop options, e.g., a higher layer of the software stack detects the unused power on a component and redistributes it to others.

Figure 3 illustrates this feature with different components. For CPUs and DRAM memories, Intel's RAPL interface [3] can be used to enforce the power cap as well as to sense the power cunsumption. Recent GPUs also support such functionalities (e.g., nvidia-smi interface

for NVIDIA GPUs [4]). Cooling facilities generally do not support the power capping features, and a software layer to control the power is needed to include them for the power budgeting loop. We can trade the power budgets across different hardware by using the interfaces, i.e., setting power caps to them accordingly and adjusting them depending on their demands detected by use of the measured power. In this document, we call this control feature a *power cap knob* as a subset of general power management knobs. In addition to the power cap knob, as an option, we also consider directly controlling the hardware knobs to trade off power and performance for some use cases while keeping the power cap constraint (e.g., minimizing energy while keeping the power constraint).

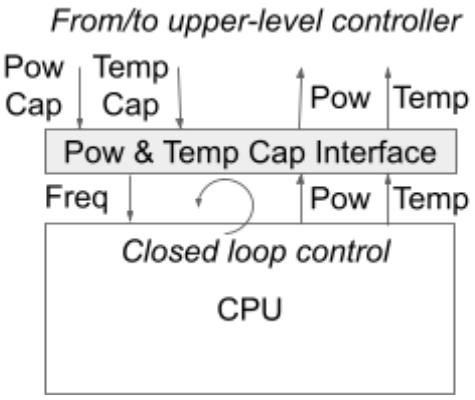

Figure 4: Interface Extension for Thermal Capping

A more advanced option for the interface is augmenting a thermal capping functionality. Figure 4 illustrates an example with CPU. Similar to the power capping, to enable the thermal capping, the interface should support a closed-loop controlling using the power management knob(s), including any power-performance knobs such as clock scaling and the power cap knob, while monitoring the temperature so that it doesn't exceed the designated temperature limit. The interface should also be able to report the temperature so that we can double check the correctness even if we never exchange the temperature across nodes/jobs/components. In this document, we call this temperature control feature a *thermal cap knob*.

The thermal capping feature is widely supported in modern commercial processors. They usually have a temperature controlling hardware to prevent overheating, and if the temperature exceeds a predetermined threshold (usually set very high such as 80°C), the hardware module attempts to throttle the throughput by scaling down the clock frequency, thinning out the clocks, or any other throttling mechanisms. The thermal threshold is exposed to the operating system layer for several processors [5]. If the threshold is constant, which is the case for some commercial processors, the temperature capping should be implemented in a low-level software layer using the mechanism shown in Figure 4. This functionality can be implemented inside of the Node Manager tools, such as PULP Controller and BEO.

4.2 Use Case / Requirements Description Format

In this section, we define the description formats of use cases and their requirements in a general and comprehensive manner, following the open standard philosophy of our REGALE architecture. A power control optimization should be governed by a certain optimization problem (e.g., maximizing total system throughput under a power constraint), and thus we define a use case in a form of an optimization problem while covering the following aspects:

1. **Optimization objective(s) for power control**
2. **Constraint(s) for power control**
3. **Control knob(s)**
4. **Temporal/spatial granularity**
5. **Optimizer(s)**

Optimization objective(s) for power control specifies the objective function(s) to optimize the power control hardware. This can be a system-focused objective (e.g., maximizing total system throughput or energy-efficiency) or an application-focused one (e.g., minimizing application runtime or energy consumption). We can cover multi-objective optimizations as well by setting the objective function as a linear sum (or any other arithmetic function) of these different objectives and setting the weights (or coefficients) accordingly. As an extreme case, the most naive one does not have any objective functions, but just sets power management knobs to functionally enforce constraints without any optimizations.

Constraint(s) for power control is one (or more) constraint(s) set when controlling power management knobs. This could be power, thermal, performance or any other constraints (including anomaly tolerability/detectability) at the entire system or application level, depending on the use case. In case there is no optimization objective, only the constraints affect the actual power management knob setups. For instance, we attempt to maximize total system throughput under given power and thermal constraints in a use case by optimizing the power management knob setups. In case, we have no objective function, we only enforce the limits but do not optimize anything.

Control knob(s) specifies what hardware/software knobs we control to achieve the objective(s) while keeping the constraint(s) in the use case. This includes any kind of power management knobs including in-band power controllers in CPUs, GPUs, or DRAM memories (e.g., RAPL-based power capping or clock scaling features) and out-of-band ones, such as the temperature controller inside of a cooling facility. Further, we also cover other means to optimize power/energy, such as job scheduling and power/energy-aware code optimizations. Some use cases highly rely on the control knobs (e.g., need the power cap knob), and thus applicable use cases to a system highly depend on the available knobs as well.

Temporal/spatial granularity of power control determines when/where the optimization is applied. The temporal granularity decides the timing of when the power knob setup changes, such as only at a job launch or in a periodical manner with a certain interval. The spatial granularity specifies where the optimization happens, such as within each compute node, across different compute nodes/jobs, or at an even coarser level, e.g., across the entire compute nodes, the entire I/O nodes, and the cooling facility. The power control decision would be hierarchical when combining them.

Optimizer(s) defines at which level the power control optimization is applied. This can be at system level, at application level, or the combination of both. There is no optimizer if the power constraint is manually set by the site administrator.

On the other hand, by following the general REGALE architecture shown, we specify the requirements for all the components/actors per use case. More specifically, we cover the following components/actors in the specifications here. Note that we will consider adding the workflow engine to cover more use cases when integrating the different paths in future work.

1. **Site Admin**
2. **Users/Developers**
3. **System Manager (RJMS/SPM)**
4. **Job Manager**
5. **Node Manager**
6. **Monitor (Dashboard/Signature Handler/Estimator)**

Site Admin can have some roles in several use cases. One example is interacting with the system for a certain setup (e.g., total system power constraint), and another one is fixing their policy such as token consumption accounting, i.e., the rules on how much they charge for a job, so that it fits well the given use case. For instance, some of the user-level optimizations should pay incentives for users otherwise they are not profitable for users (but at least beneficial for the site admin).

Users/Developers also can have some roles for optimization as well. As an example, for a user-level power or energy optimizations, they might need to link some relevant libraries to optimize their code. Another example is that setting up some environmental variables in their job scripts could be required to enable some features.

System Manager (including RJMS and SPM), Job Manager, Node Manager, **Monitor** are the software components included in the REGALE architecture (see Figure 1). Some use cases need to coordinate all of them while others may need only some of them. The requirements highly depend on all the aspects that determine the use cases (objectives, constraints, etc).

4.3 Requirement Specifications per Use Case

Before selecting and defining use cases, we consider different **levels of sophistication** for different aspects as shown in TABLE 2. By designating the level for each aspect, we can define a use case. As for objectives, we can increase the number of objective functions to consider, and we will work on multi-objective optimizations in WP2. For constraints, we will start from a single constraint (e.g., power) and then later explore multiple constraints (e.g., power and temperature constraints) as well. For the temporal granularity, we can both cover static and dynamic manners. As for the spatial granularity, the initial step is setting the node power management knobs evenly across different nodes, and we gradually include optimizations at different levels. As for the knobs for optimizations, we first target only the CPU power management knob, then cover multiple different components (e.g., GPUs and memories), and finally include job scheduler-level power and energy optimizations.

TABLE 2: Different levels of sophistication

	Level 1	Level 2	Level 3	Level 4
Objective(s)	None	One (system or app focused)	Multiple objectives	. . .

Constraint(s)	One (e.g., power constraint)	Two (e.g., power + temperature)	More (e.g., power + temp + anomaly detectable)	...
Temporal Granularity	Statically set by site admin	Statically set when job launch	Dynamically adjusted at runtime	...
Spatial Granularity	Entire compute nodes (all nodes work uniformly)	Intra- xor inter-node optimization	Both intra- and inter-node optimization	Include other kinds of nodes or facilities
Knob(s)	CPU power mgmt knob	Power mgmt knobs of multiple components	Include job scheduling optimizations	...
Optimizer(s)	None	One (system xor user level optimization)	Both system and user level optimization	...

4.3.1 Basic Use Cases

We start from the most naive use case, i.e., all Level 1 options in the table, and the requirements are shown in TABLE 3. In this use case, we set the power limit to the entire set of compute nodes. The allocated power budget is distributed evenly across the nodes without any optimizations. We do not have any objective function here, but only try to enforce the power limit. The power control is conducted in an open-loop manner, i.e., we do not use any feedback from the measurement side, and the power capping interface is responsible for keeping the constraint.

TABLE 3: Requirement Specifications for Basic Power Capping (Basic)

Use case	Definition	Requirements
[**Basic**] Keep my system under power cap **Note:** Providing system-level power capping functionality w/o any optimizations; power budget is distributed evenly across nodes; Open-loop control w/o using measured power at runtime	**Objectives:** None (Level1) **Constraints:** Power (Level1) **Knobs:** CPU power cap (Level1) **Temporal Granularity:** Updated only when the site admin changes the setup (Level1) **Spatial Granularity:** Entire compute node (Level1) **Optimizers:** None (Level1)	**Site Admin:** Set power cap to System Manager (e.g., 1MW) **Users/Developers:** None **System Manager (SPM):** Capability/interface to talk to each node manager to set power cap to them; HW profiling functionality to obtain the range of power consumption when scaling the target knob; Report if the power budget setup is outside of the range or if a significant power budget violation happens **Node Manager:** Talk to HW and set up power cap based on the instruction by the system manager; report if an error/anomaly happens to the system manager **Job Manager:** None **Monitor:** None

We then go one step further in terms of the constraints set up. More specifically, we augment the temperature capping functionality to Basic – here, we call this use case as Basic+. Note

that to apply this thermal capping along with the power capping, we need a proper interface as described in Section 4.1. TABLE 4 summarizes the requirements to support this use case. Aside from the necessity for the temperature capping interface, the requirements are almost the same as those of Basic.

TABLE 4: Requirement Specifications for Basic Power and Thermal Capping (Basic+)

Use case	Definition	Requirements
[**Basic+**] Keep my system under power and thermal caps **Note:** Providing system-level power and thermal capping functionality w/o any optimizations; power budget is distributed evenly; the same temperature setup for every node; Open-loop control w/o using measured power nor temperature at runtime	**Objectives:** None (Level1) **Constraints:** Power and temperature (Level2) **Knobs:** CPU power & thermal caps (Level1) **Temporal Granularity:** Updated only when the site admin changes the setup (Level1) **Spatial Granularity:** Entire compute node (Level1) **Optimizers:** None (Level1)	**Site Admin:** Set power and thermal caps to System Manager (e.g., 1MW & 50°C) **Users/Developers:** None **System Manager (SPM):** Capability/interface to talk to each node manager to set power and temperature caps to them; HW profiling functionality to obtain the range of power consumption when scaling the target knob; Report if the power budget setup is outside of the range or if a significant power budget violation happens **Node Manager:** Talk to HW and set up power and thermal caps based on the instruction by the system manager; report if an error/anomaly happens to the system manager **Job Manager:** None **Monitor:** None

In TABLE 5, we extend Basic+ by adding the anomaly detectability requirement, which we call Basic++ here. If a target hardware region violates the power or thermal limits more than a certain threshold longer than a predetermined duration, this should be reported. Here, we just consider the detection and report functions, but in the future deliverables, we will cover more sophisticated options such as an anomaly tolerance option with automatic anomaly handling methodologies. An anomaly can happen at a variety of granularity levels, and thus anomalies should be detectable at all the software components. In the future work, the anomaly detection, correction, and mitigation should be realized at different levels in a hierarchical manner: (1) application; (2) subsystem; (3) node; and (4) room level. We can consider a variety of use cases even only on anomaly handling methodologies for different scenarios or target hardware.

TABLE 5: Requirement Specifications for Basic Power and Thermal Capping with Anomaly Detectability (Basic++)

Use case	Definition	Requirements

[**Basic++**] Keep my system under power and thermal caps with anomaly detectability **Note:** Providing system-level power and thermal capping functionality w/o any optimizations; power budget is distributed evenly; the same temperature setup for every node; Open-loop control w/o using measured power nor temperature at runtime; Anomaly is detectable at any components	**Objectives:** None (Level1) **Constraints:** Power and temperature constraints + anomaly detectable (Level3) **Knobs:** CPU power & thermal caps (Level1) **Temporal Granularity:** Updated only when the site admin changes the setup (Level1) **Spatial Granularity:** Entire compute node (Level1) **Optimizers:** None (Level1)	**Site Admin:** Set power and thermal caps to System Manager (e.g., 1MW & 50°C); Handle anomaly node reported by System Manager **Users/Developers:** None **System Manager (SPM):** Capability/interface to talk to each node manager to set power and temperature caps to them; HW profiling functionality to obtain the range of power consumption when scaling the target knob; Report if the power budget setup is outside of the range or if a significant power budget violation happens; Anomaly detection function in terms of power and temperature (reported by other components); Report when anomaly is detected to site admin **Node Manager:** Talk to HW and set up power and thermal caps based on the instruction by the system manager; report if an error/anomaly happens to the system manager **Job Manager:** Report if an error/anomaly happens to the system manager **Monitor:** Provides information on facility and nodes anomalies

4.3.2 Advanced Use Cases

Next, we extend the Basic use case by optimizing the hardware power management knob setup while following a given objective function. Here, we cover the following objectives: maximizing total system throughput (SysThru); minimizing total system energy (SysEne); maximizing application performance (AppPerf); and minimizing application energy-to-solution (AppEtS). For all of these use cases, we assume the site administrator sets the power

constraint to the entire set of compute nodes, and then we optimize the power budgeting across these nodes while keeping the constraint to achieve a given objective. TABLE 6 describes the definition/requirements for each of these options. Here, we consider power is only the constraint, however this can be extended to cover more constraints by adding requirements listed in Basic+ or Basic++. Another option for the constraints is considering average or maximum application performance degradation. For SysThru, we consider closed-loop power controls in these use cases, i.e., we dynamically adjust the power management knob in accordance with the measured power consumption and resource utilizations at runtime. These measurements are used for estimating the power demand of a node by the Node Manager, which is then sent to the System Manager to redistribute the power budgets across nodes. SysEne is almost the same as SysThru except for the objective function and an option to scale down the total power budget allocated to the entire set of compute nodes, which could improve energy efficiency but wouldn't improve throughput. On the other hand, AppPerf and AppEtS are application level (or user level) optimizations. In these use cases, we utilize application profiles of previous or test runs, which are provided by Monitor. By analyzing the profiles, Job Manager decides the setups of the target power management knob (CPU power cap, CPU clock frequency scaling or any others) as well as performs code tuning as an option. One needs to link the specific libraries to the code to realize these application level options.

TABLE 6: Requirement Specifications for Optimization Variants

Use case	Definition	Requirements
[**SysThru**] Maximizing total system throughput under power cap **Note:** Optimize power budget allocations across nodes under the total system power cap so that the total system throughput can be maximized; Closed-loop power management at runtime; Assuming over provisioned situation; Adding more constraints is an option (e,g., thermal cap, application speed-down limit)	**Objectives:** Max system throughput (Level2) **Constraints:** Power cap (Level1) – or extensible to include more (Level2/3) **Knobs:** CPU power cap (Level1) **Temporal Granularity:** Statistically set at runtime (Level2) or dynamically adjusted at runtime (Level3) **Spatial Granularity:** Inter node optimization (Level2) **Optimizers:** System-level optimization (Level1)	**Site Admin:** Set power cap to the entire compute nodes via System Manager (e.g., 1MW); Revisit the token accounting policy to deal with potential unfairness **Users/Developers:** None **System Manager (SPM):** All the functionalities supported in Basic; A profile-based power budget decision making (static); Periodical power budget redistribution function based on the reported unused power and power budget request by Node Manager (dynamic); Power budget distribution algorithm to maximize throughput **Node Manager:** All the functionalities supported in Basic; Policy to detect whether the node needs less/more power budget and report it to System

		Manager (dynamic) **Job Manager:** None **Monitor:** Providing monitoring data to SPM and Node Manager when dynamically adjust the power cap; Providing power/performance estimation to other actors
[**SysEne**] Minimizing total system energy consumption under power cap **Note:** The requirements are almost the same as those for SysThru; Need to update the power distribution policy/algorithm from SysThru, in particular Node Manager level; Adding more constraints is an option (e,g., thermal cap, application speed-down limit)	**Objectives:** Min system energy consumption (Level2) **Constraints:** Power cap (Level1) – or extensible to include more (Level2/3) **Knobs:** CPU power cap (Level1) **Temporal Granularity:** Statistically set at runtime (Level2) or dynamically adjusted at runtime (Level3) **Spatial Granularity:** Inter node optimization (Level2) **Optimizers:** System-level optimization (Level1)	**Site Admin:** Same as SysThru **Users/Developers:** None **System Manager (SPM):** Same as SysThru; Scaling down total system cap adaptively is an option **Node Manager:** Same as SysThru but need updates in the power budget request policy, i.e., detecting the optimal power mgmt knob setup to minimize energy (or maximize energy efficiency) **Job Manager:** None **Monitor:** Providing monitoring data to Node Manager when dynamically adjust the power cap; Providing power/performance/energy estimation to other actors
[**AppPerf**] Maximizing application performance under power cap **Note:** Similar to SysThru, but allows users to optimize power knobs while keeping power cap; Adding more constraints is an option (e,g., thermal cap)	**Objectives:** Max system throughput (Level2) **Constraints:** Power cap (Level1) – or extensible to include more (Level2/3) **Knobs:** CPU power mnmt knob (Level1) **Temporal Granularity:** Statically set when job launch (Level2) or Dynamically adjusted at runtime (Level3) **Spatial Granularity:** Inter node optimization (Level2)	**Site Admin:** Same as SysThru; Allow user level (or Job Manager level) power management **Users/Developers:** Link the relevant library (provided by Job Manager) to their code **System Manager:** Same as SysThru **Node Manager:** Same as SysThru except that it needs to provide an interface to let Job Manager know the current power knobs and allow it to further optimize them

	Optimizers: App-level optimization (Level1)	**Job Manager:** Power-aware code tuning functionality using code analysis or profile-based optimization; Accessible power management knobs (power cap, clock freq, etc.) **Monitor:** Providing monitored stats to Job Manager.
[**AppEtS**] Maximizing energy to solution for app under power cap **Note:** Almost same as AppPerf except that the objective is minimizing energy; Adding more constraints is an option (e,g., thermal cap, application speed-down limit)	**Objectives:** Max system throughput (Level2) **Constraints:** Power cap (Level1) – or extensible to include more (Level2/3) **Knobs:** CPU power mgmt knob (Level1) **Temporal Granularity:** Statically set when job launch (Level2) or Dynamically adjusted at runtime (Level3) **Spatial Granularity:** Inter node optimization (Level2) **Optimizers:** App-level optimization (Level1)	**Site Admin:** Same as AppPerf **Users/Developers:** Same as AppPerf **System Manager:** Same as AppPerf **Node Manager:** Same as AppPerf **Job Manager:** Same as AppPerf **Monitor:** Same as AppPerf except that the optimization policy must be updated.

4.3.3 More Advanced Use Cases

We then cover more advanced options by extending one of the above optimization variants. We focus on SysThu as an example, but the requirements here are general and should stand regardless of the objective function setup while who optimizes can be different. First, we consider NodPowShift: power shifting across different components inside of a node. To support this option, the most significant modification would be in Node Manager, i.e., extending the existing knob control policy and providing the functionality to distribute power budgeting among in-node components. Second, as a next step, we consider a use case named SchedOpt: power-aware job scheduling support along with power management knob optimizations. This use case requires application characteristic analysis using historical data

collected by Monitor, regarding throughput, energy efficiency, and so forth under a given power control scheme. The requirements for these two use cases are specified in TABLE 7.

TABLE 7: Requirement Specifications for Advanced Use Cases

Use case	Definition	Requirements
[**NodPowShft**] Maximizing a given objective function under one or more constraints via coordinating knobs of different in-node components **Note:** Should be agnostic in objectives and constraints except for who manages it and the actual policy	**Objectives:** Max system throughput or min system energy (Level2) **Constraints:** Power cap (Level1) – or extensible to include more (Level2/3) **Knobs:** CPU power cap + other components like GPU or DRAM memory power caps (Level2) **Temporal Granularity:** Statistically set at runtime (Level2) or dynamically adjusted at runtime (Level3) **Spatial Granularity:** Inter- and intra-node optimization (Level3) **Optimizers:** App- or System-level optimization (Level1) or both (Level2)	**Site Admin:** Same as SysThru **Users/Developers:** Same as SysThru **System Manager:** Same as SysThru **Node Manager:** Needs a layer to distribute a power cap to different component; Update the policy to exploit the above feature **Job Manager:** Same as SysThru **Monitor:** Same as SysThru
[**SchedOpt**] Maximizing total system throughput under power cap w/ job scheduling optimization (+ intra-node power shifting) **Note:** Should be agnostic in objectives and constraints; Needs job characteristics estimation using historical data collected by Monitor	**Objectives:** Max system throughput (Level2) **Constraints:** Power cap (Level1) – or extensible to include more (Level2/3) **Knobs:** CPU power cap + other components like GPU or DRAM memory power caps + job scheduling (Level3) **Temporal Granularity:** Dynamically adjusted at runtime (Level3) **Spatial Granularity:** Inter- and intra-node optimization (Level3)	**Site Admin:** Same as NodPowShft **Users/Developers:** Same as NodPowShft **System Manager (RJMS):** Power-aware scheduling policy using the historical job statistics; Accessible to system power status such as remaining power budget on the system **Node Manager:** Same as NodPowShft **Job Manager:** Same as NodPowShft

	Optimizers: App- or System-level optimization (Level1) or both (Level2)	**Monitor:** Providing collected job statistics under power optimizations

4.3.4 Discussions

In summary, we picked and defined the classes and use cases listed in TABLE 8. Some of these use cases are aligned with the PowerStack initiative community [1]. Our major contribution here is describing them in a generalized form while defining the levels of sophistication in different aspects shown in TABLE 2.

TABLE 8: Summary of classes and use cases

Class	Code name
Basic	Basic, Basic+, Basic++
Advanced	SysThru, SysEne, AppPerf, AppAtS
More Advanced	NodPowShft, SchedOpt

We will continuously assess and update the use cases in terms of necessity, coverage, comprehensiveness, and so forth throughout the project. Our goal of this work package is *not* defining use cases and architecture in order to integrate software in WP3, but is rather comprehensive. Our ultimate goal here is designing an open standard architecture, while reflecting the community-wide trends/opinions including outside of the project such as the PowerStack initiatives [1]. For this reason, we will cover use cases and architectural modules that may not be implemented in WP3. However, they will work as a milestone for any current/future software implementations to realize power-aware HPC. In WP3, as an example, we picked up a set of use cases relevant to our software tools and are integrating them to realize the use cases. Further, we will continuously apply minor updates to our use cases and architecture based on the practices experienced in WP3.

In the rest of the deliverable, we describe the necessary functionalities/interfaces to realize these use cases in Section 5. Based on the detailed architecture descriptions, we assessed the current state of our software tools, decided software integration instances (or scenarios), and then started integrating our tools to realize the use cases from naive to sophisticated. Section 6 summarizes the current status of our integrations as well as the current coverages of these use cases. Further, we also introduce more sophisticated use cases in the final deliverables based on the ongoing studies conducted in WP2.

5. Architecture and Interface Descriptions

In this section, we explain the details of the current architecture with a particular focus on the PowerStack path. More specifically, by following our intermediate architecture described in Section3 (see also Figure 1) as well as the PowerStack use cases and their requirements clarified in Section4, we define the details of the Regale open architecture while providing a necessary and sufficient set of interface functions to realize the PowerStack use cases defined in the previous section. The interface functions in the real world can be implemented as APIs, command line interfaces provided by services/modules, database queries, configuration files, and so forth, depending on the functionalities, the details of which are provided in the WP3 deliverable. Here, we describe them in a generic way consisting of input/output variables. In this section, we firstly present the functions to realize the Basic Use Cases (Basic/Basic+/Basic++) and then secondly describe the rest of the functions for the rest of the advanced use cases. We then finally present how these use cases work using these interface functions. In the final deliverable, the interface descriptions will cover also the Melissa and RYAX paths, which are used for realizing sophisticated HPC workflow managements. Note their initial architectures are introduced in the previous version of our deliverable (D1.1).

The following text in the deliverable provides a roadmap of the high-level interfaces so that the modules/actors in the REGALE architecture interoperate for different use cases. The naming convention has been generated following a top-down approach. D3.1 followed a bottom-up approach defining the list of interfaces needed by the tools. In the next period, we will implement a common naming convention between the final architecture specification and tools' interface implementation.

5.1. Interface Functions for Basic Use Cases

Figure 5 depicts the overall architecture and the information flows among the software components with a particular focus on the basic use cases. The system administrator configures and selects the policy of the resource management and also designates the system-wide power (or temperature) cap(s) for the basic power/thermal management. The system administrator can add/remove a node to/from the power management target, based on such as the anomaly detected by the Estimator.

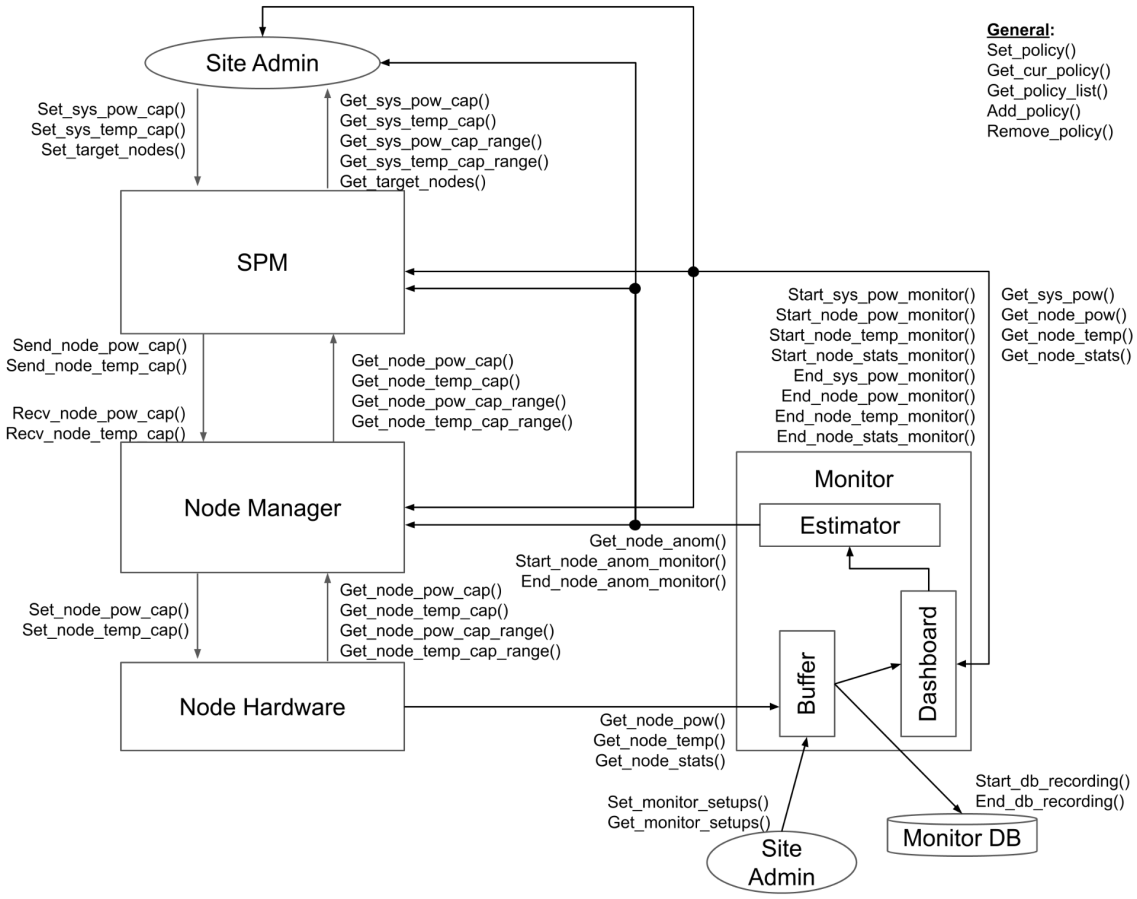

Figure 5: Overview of Architecture and Information Flow for Basic Use Cases

[Component Agnostic]

The following interface functions are needed for the site administrator to select/configure the management policy:

Set_policy(Policy, Status)

IN Policy	A power management policy as an instance of a use case
OUT Status	List of errors observed during the procedure

Get_cur_policy(Policy, Status)

OUT Policy	A power management policy as an instance of a use case
OUT Status	List of errors observed during the procedure

Get_policy_list(Policy_list, Status)

OUT Policy_list	List of power management policies available on this system
OUT Status	List of errors observed during the procedure

These functions are exposed to the site administrator so as to designate a power management policy (Policy) from a set of policies (Policy_list) derived from the use cases. Once the policy setup is changed, the change is applied by selecting the associated configuration or plugin. The output variable Status returns the acknowledgement or error type to report if the policy setup completes properly or not. The interface could be provided by

each component, or one software layer could govern the policy setup across different components for the synchronization purpose.

The site administrator should be able to add/remove/modify a policy freely. To this end, the following interface should also be provided. The function Add_policy is used to install a new policy with the policy setup input stated as Policy_config, which could be a configuration file or a plugin depending on the implementation.

Add_policy(Policy_config, Status)

IN	Policy_config	A new power management policy to install on this system
OUT	Status	List of errors observed during the procedure

Remove_policy(Policy, Status)

IN	Policy	A power management policy to remove from this system
OUT	Status	List of errors observed during the procedure

[System Manager (SPM)]

Here, we describe the interfaces provided by the System Manager. The RJMS is not involved in the power management for the basic use cases, the interfaces below are only relevant to the SPM.

To Site Administrator

The following three functions are used to set/check the total power cap set to the entire system. These interfaces are used also for any other advanced use case when applying the total system power constraint. The use case Basic++ is able to detect anomalies, and thus they also report the detailed anomaly status if detected, using a variable Status. The site administrator picks up a total power cap within the power cap range provided by the third function.

Set_sys_pow_cap(Sys_pow_cap, Status)

IN Sys_pow_cap		Total system power cap set by the site administrator
OUT Status		List of errors observed during the procedure

Get_sys_pow_cap(Sys_pow_cap, Status)

OUT Sys_pow_cap		Current total system power cap set by the site admin
OUT Status		List of errors observed during the procedure

Get_sys_pow_cap_range(Min_sys_pow_cap, Max_sys_pow_cap)

OUT Min_sys_pow_cap		Minimum total system power cap
OUT Max_sys_pow_cap		Maximum total system power cap

Similarly, the following three functions are used to set/check the system-wide temperature constraint setups. This set of interface functions are the same as the above system-wide power cap functions except that the target is temperature.

Set_sys_temp_cap(Sys_temp_cap, Status)

IN Sys_temp_cap		Temperature cap set to all nodes by the site admin
OUT Status		List of errors observed during the procedure

Get_sys_temp_cap(Sys_temp_cap, Status)

OUT	Sys_temp_cap	Current system-wide temperature capping setup
OUT	Status	List of errors observed during the procedure

Get_sys_temp_cap_range(Min_sys_pow_cap, Max_sys_pow_cap)

OUT	Min_sys_temp_cap	Minimum value for Sys_temp_cap
OUT	Max_sys_temp_cap	Maximum value for Sys_temp_cap

The following functions are used to configure the power management target, manually by the site administrator. The Node_config could be a configuration file that specifies all the power/thermal management target nodes with associated information, while the Node_list lists up all the power management targets. The site administrator can manually exclude an anomaly node from the target and include it once the anomaly status is solved.

Set_target_nodes(Node_config, Status)

IN	Node_config	Target nodes to apply power/thermal management
OUT	Status	List of errors observed during the procedure

Get_target_nodes(Node_list, Status)

OUT	Node_list	Current power/thermal management targets
OUT	Status	List of errors observed during the procedure

To Node Manager

The above functions are rather for the site administrator, while the following interfaces are used to interact with the node manager. In particular, the following two functionalities are used to set the power/thermal cap to the node selected with the ID (Node_id). The nodes here include compute nodes, I/O nodes, or any kinds of system components. The variable Status represents if the setup procedure completes successfully or not.

Send_node_pow_cap(Node_id, Node_pow_cap, Status)

IN	Node_id	Node id to specify the target
IN	Node_pow_cap	Power cap to set to the designated node
OUT	Status	List of errors observed during the procedure

Send_node_temp_cap(Node_id, Node_temp_cap, Status)

IN	Node_id	Node id to specify the target
IN	Node_temp_cap	Node thermal cap set to the designated node
OUT	Status	List of errors observed during the procedure

The functions below are used by the node manager to snoop the power/thermal cap request.

Recv_node_pow_cap(Node_pow_cap, Status)

OUT	Node_pow_cap	Power cap to set to the designated node
OUT	Status	List of errors observed during the procedure

Recv_node_temp_cap(Node_temp_cap, Statu)

OUT	Node_temp_cap	Thermal cap to set to the designated node
OUT	Status	List of errors observed during the procedure

[Node Manager]

The following interface functions are provided by the node manager to interact with the system manager, in particular, the SPM.

To System Manager (SPM)

These two functions are used to send the current power/thermal cap information to the system manager side.

Get_node_pow_cap(Node_id, Node_pow_cap, Status)

IN	Node_id	id of this node
OUT	Node_pow_cap	Current node power cap set at this node
OUT	Status	List of errors observed during the procedure

Get_node_temp_cap(Node_id, Node_temp_cap, Status)

IN	Node_id	id of this node
OUT	Node_temp_cap	Current node power cap set at this node
OUT	Status	List of errors observed during the procedure

The following two functions are used to report the node power or thermal cap range to the system manager after probing the hardware.

Get_node_pow_cap_range(Node_id, Min_node_pow_cap, Max_node_pow_cap)

IN	Node_id	id of this node
OUT	Min_node_pow_cap	Minimum power cap of this node
OUT	Max_node_pow_cap	Maximum power cap of this node

Get_node_temp_cap_range(Node_id, Min_node_temp_cap, Max_node_temp_cap)

IN	Node_id	id of this node
OUT	Min_sys_temp_cap	Minimum temp cap of this node
OUT	Max_sys_temp_cap	Maximum temp cap of this node

To Hardware Modules

The following interface functions are used to interact with the hardware modules (or low level software), in particular at the granularity of nodes.

Set_node_pow_cap(Node_pow_cap, Status)

IN	Node_pow_cap	Node power cap to set at this node
OUT	Status	List of errors observed during the procedure

Get_node_pow_cap(Node_pow_cap, Status)

OUT	Node_pow_cap	Current power cap set at this node
OUT	Status	List of errors observed during the procedure

Get_node_pow_cap_range(Min_node_pow_cap, Max_node_pow_cap)

OUT	Min_sys_pow_cap	Minimum power cap of this node
OUT	Max_sys_pow_cap	Maximum power cap of this node

Set_node_temp_cap(Node_temp_cap, Status)

IN	Node_temp_cap	Node thermal cap to set at this node
OUT	Status	List of errors observed during the procedure

Get_node_temp_cap(Node_temp_cap, Status)

OUT	Node_temp_cap	Current thermal cap set at this node
OUT	Status	List of errors observed during the procedure

Get_node_temp_cap_range(Min_node_temp_cap, Max_node_temp_cap)

OUT	Min_sys_temp_cap	Minimum thermal cap of this node
OUT	Max_sys_temp_cap	Maximum thermal cap of this node

[Monitor]

The following functions are used to set up the monitor configurations, in particular specifying what to monitor with per-sensor options, such as sampling frequency.

Set_monitor_setups(Monitor_config, Status)

IN	Monitor_config	This input specifies monitoring configurations, including the target sensors/monitors, their locations, monitoring frequency, etc.
OUT	Status	List of errors observed during the procedure

Get_monitor_setups(Monitor_config, Status)

OUT	Monitor_config	A list to specify current monitoring configurations, including the target sensors/monitors, their locations, monitoring frequency, etc.
OUT	Status	List of errors observed during the procedure

Monitor DB

The following functions are used to start/stop recording the monitored statistics on the database. The monitoring targets are specified by the variable Targets, and if not specified, all the sensors/counters configured in the above monitor setup functions are recorded.

Start_db_recording(Duration, Push_freq, Targets, DB, Status)

IN	Duration	A variable to specify the duration of recording
IN	Push_freq	Frequency to push the data to the database
IN	Targets	A list to specify target sensors/counters to record
IN	DB	ID to designate the database
OUT	Status	List of errors observed during the procedure

The following function is used for stop recording.

End_db_recording(Status)

OUT	Status	List of errors observed during the ending procedure

Dashboard

The dashboard is used for displaying the monitored data to any actors including both human actors and software components. In particular, the following functions are used to obtain power, temperature, or any other statistics at a certain timing. The function Get_sys_power()

returns the system-wide power consumption, which could be the summation of power consumption of all nodes or based on a dedicated power measurement facility.

Get_sys_pow(Sys_pow, Time, Status)

OUT	Sys_pow	Current total system power consumption
OUT	Time	Time stamp of the measurement
OUT	Status	List of errors observed during the procedure

Get_node_pow(Node_id, Node_pow, Time, Status)

IN	Node_id	Node id to specify the target
OUT	Node_pow	Current power consumption at this node
OUT	Time	Time stamp of the measurement
OUT	Status	List of errors observed during the procedure

Get_node_temp(Node_id, Node_temp, Time, Status)

IN	Node_id	Node id to specify the target
OUT	Node_temp	Current temperature at this node
OUT	Time	Time stamp of the measurement
OUT	Status	List of errors observed during the procedure

Get_node_stats(Node_id, Targets, Stats, Time, Status)

IN	Node_id	Node id to specify the target node
IN	Targets	A list to specify target sensors/counters belongs to the node
OUT	Stats	List of statistics provided by the sensors/counters in the node
OUT	Time	Time stamp of the measurement
OUT	Status	List of errors observed during the procedure

The following functions are used to start/end periodically obtaining monitored data with a specified monitoring frequency.

Start_monitoring_node_pow(Node_id, Duration, Sens_freq, Recv_freq, Node_pow, Status)

IN	Node_id	Node id to specify the target
IN	Duration	A variable to set monitoring duration
IN	Sens_freq	A variable to set monitoring frequency
IN	Recv_freq	A variable to set the frequency of receiving data
OUT	Node_pow	Time series data of power consumption at this node
OUT	Status	List of errors observed during the procedure

Start_monitoring_node_temp(Node_id, Duration, Sens_freq, Recv_freq, Node_temp, Status)

IN	Node_id	Node id to specify the target
IN	Duration	A variable to set monitoring duration
IN	Sens_freq	A variable to set monitoring frequency
IN	Recv_freq	A variable to set the frequency of receiving data
OUT	Node_temp	Time series data of temperature at this node
OUT	Status	List of errors observed during the procedure

Start_monitoring_node_stats(Node_id, Duration, Sens_freq, Recv_freq, Targets, Stats, Status)

IN	Node_id	Node id to specify the target node

IN	Duration	A variable to set monitoring duration
IN	Sens_freq	A variable to set monitoring frequency
IN	Recv_freq	A variable to set the frequency of receiving data
IN	Targets	A list to specify target sensors/counters belongs to the node
OUT	Stats	A time series data of statistics obtained from the target sensors/counters in the node
OUT	Status	List of errors observed during the procedure

End_monitoring_node_pow(Status)

OUT	Status	List of errors observed during monitoring and the end procedure

End_monitoring_node_temp(Status)

OUT	Status	List of errors observed during monitoring and the end procedure

End_monitoring_node_stats(Status)

OUT	Status	List of errors observed during monitoring and the end procedure

Estimator

The estimator inside the monitor tool continuously collects the node statistics and broadcasts any detected anomalies to any actors including both the human actors and the software components. The detectable anomalies can differ depending on the implementations and should be implemented as plugins for the extensibility.

The following function checks anomalies at a certain timing in the target node. It returns errors if any anomalies are detected. It detects the thermal/power cap violations, and other anomalies using the statistics collected on the target node.

Get_node_anom(Node_id, Node_pow_cap, Node_temp_cap, Node_pow, Node_temp, Stats, Status)

IN	Node_id	Node id to specify the target node
IN	Node_pow_cap	Current node power cap set at this node
IN	Node_temp_cap	Current node power cap set at this node
IN	Node_pow	Time series data of actual power consumption at this node
IN	Node_temp	Time series data of actual temperature at this node
IN	Stats	Time series data of statistics obtained from the target sensors/counters in the node
OUT	Status	List of anomalies confirmed during the procedure

The following functions are used to continuously detect the anomalies on the target node. Every time an anomaly is detected, it is listed in the output (Status).

Start_node_anom_monitor(Node_id, Duration, Node_pow_cap, Node_temp_cap, Node_pow, Node_temp, Stats, Status)

IN	Node_id	Node id to specify the target node
IN	Duration	A variable to set monitoring duration
IN	Node_pow_cap	Current node power cap set at this node
IN	Node_temp_cap	Current node power cap set at this node
IN	Node_pow	Time series data of actual power consumption at this node
IN	Node_temp	Time series data of actual temperature at this node

IN	Stats	Time series data of statistics obtained from the target sensors/counters in the node
OUT	Status	List of anomalies observed during the monitoring

End_node_anom_monitor(Node_id, Status)

IN	Node_id	Node id to specify the target node
OUT	Status	List of errors observed during the procedure

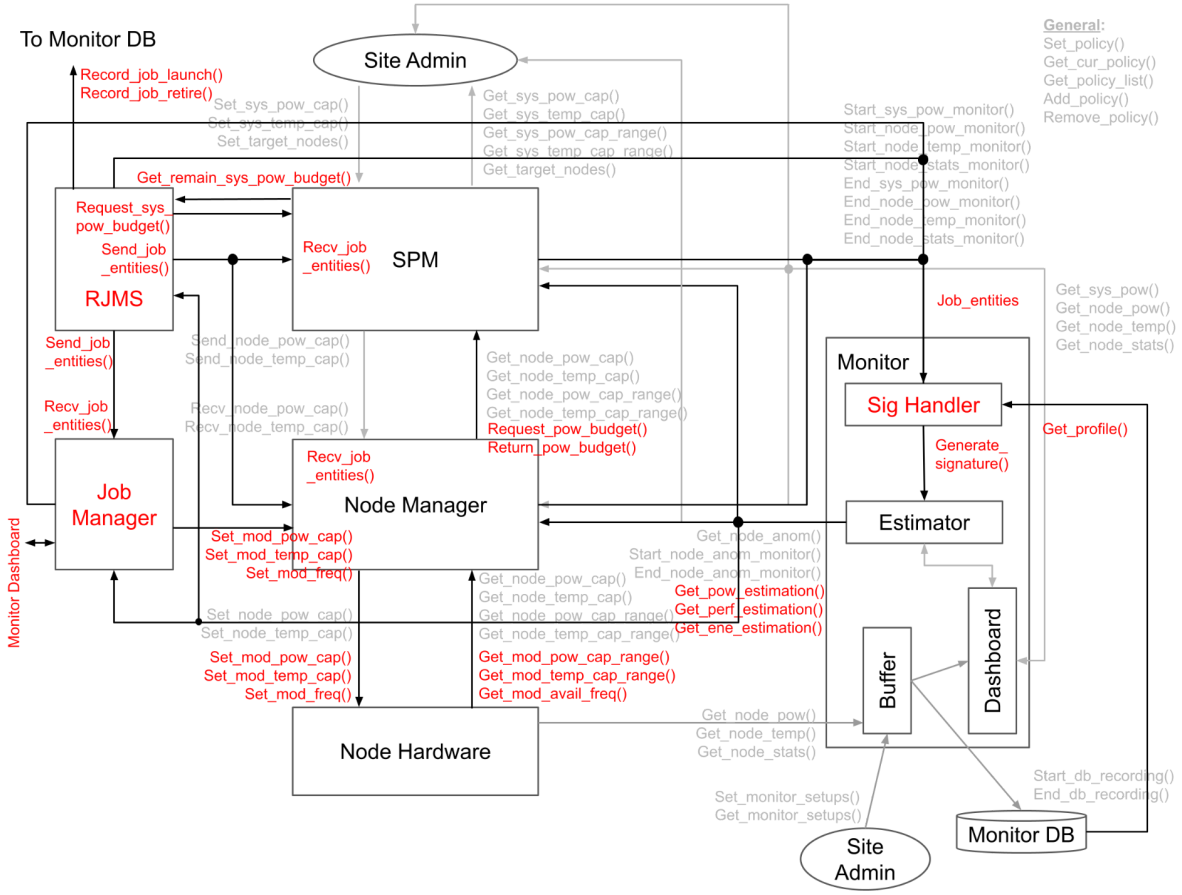

Figure 6: Overview of Architecture and Information Flow for Advanced and More Advanced Use Cases

5.2. Interface Functions for Advanced Use Cases

In this section, we describe interface functions necessary to realize the advanced use cases. Figure 6 depicts added interfaces with the entire software architecture.

[System Manager (RJMS)]

The following two functions are utilized to send out the job information to other software components in order to obtain such as associated profile or estimated performance, power, or performance (profile-based estimation). The RJMS initiates the trigger, and other components snoop the information by using the function Recv_job_entities.

Send_job_entities(Job_entities, Status)

IN	Job_entities	A set of information associated with the job (job ID, user ID,

		binary, etc.)
OUT	Status	List of errors observed during the procedure

Recv_job_entities(Job_entities, Status)

OUT	Job_entities	A set of information associated with the job (job ID, user ID, binary, etc.)
OUT	Status	List of errors observed during the procedure

[System Manager (SPM)]

The following functions are provided to mainly the RJMS to notify the remaining system power budget and to receive requests to increase the remaining system power budget. Based on them, the RJMS makes decisions on job scheduling, i.e., power- and/or energy-aware job scheduling (SchedOpt).

Get_remain_sys_pow_budget(Remain_sys_pow_budget, Status)

OUT	Remain_sys_pow_budget	Remaining system power budget
OUT	Status	List of errors observed during the procedure

Request_sys_pow_budget(Sys_pow_budget, Approval, Status)

OUT	Sys_pow_budget	Additional power budget request to raise the remaining system power budget
OUT	Approval	A variable to tell if the request is approved by the SPM or not
OUT	Status	List of errors observed during the procedure

The following functions are provided to mainly the Node manager so that it can adaptively set up the power capping depending on the characteristics of the currently running job. It requests or returns a certain amount of power budget to the SPM, and it increases or decreases the node power cap if it is approved by the SPM.

Request_node_pow_budget(Node_pow_budget, Approval, Status)

IN	Node_pow_budget	Power budget request to increase the node power cap
OUT	Approval	A variable to tell if the request is approved by the SPM or not
OUT	Status	List of errors observed during the procedure

Return_node_pow_budget(Node_pow_budget, Approval, Status)

IN	Node_pow_budget	Returning power budget to decrease the node power cap
OUT	Approval	A variable to tell if the power budget is successfully returned to the SPM or not
OUT	Status	List of errors observed during the procedure

[Node Manager]

The following functionalities are utilized to realize the module (or component) level power control inside a node, which are used in particular for power shifting across modules inside a node, i.e., NodPowShft. These functions should also be exposed to the Job Manager to conduct the application-level optimizations (AppPerf or AppEtS).

Set_mod_pow_cap(Module_id, Mod_pow_cap, Status)

IN	Module_id	Target hardware module in this node
IN	Mod_pow_cap	Power cap to set at this hardware module
OUT	Status	List of errors observed during the procedure

Set_mod_temp_cap(Module_id, Mod_temp_cap, Status)

IN	Module_id	Target hardware module in this node
IN	Mod_temp_cap	Thermal cap to set at this hardware module
OUT	Status	List of errors observed during the procedure

Set_mod_freq(Module_id, Mod_freq, Status)

IN	Module_id	Target hardware module in this node
IN	Mod_freq	Clock frequency set at this hardware module
OUT	Status	List of errors observed during the procedure

Get_mod_pow_cap(Module_id, Mod_pow_cap, Status)

IN	Module_id	Target hardware module in this node
OUT	Mod_pow_cap	Current power cap set at this hardware module
OUT	Status	List of errors observed during the procedure

Get_mod_pow_cap_range(Module_id, Min_node_pow_cap, Max_node_pow_cap)

IN	Module_id	Target hardware module in this node
OUT	Min_sys_pow_cap	Minimum power cap of this node
OUT	Max_sys_pow_cap	Maximum power cap of this node

Get_mod_temp_cap_range(Module_id, Min_node_temp_cap, Max_node_temp_cap)

IN	Module_id	Target hardware module in this node
OUT	Min_sys_temp_cap	Minimum thermal cap of this node
OUT	Max_sys_temp_cap	Maximum thermal cap of this node

Get_mod_avail_freq(Module_id, Freq_list)

IN	Module_id	Target hardware module in this node
OUT	Freq_list	List of clock frequency available at this module

[Monitor]

The following two functions are provided to the RJMS to associate the job information with the node statistics recorded on the database, i.e., what job is running on a certain node during a certain period of time. By doing so, profile-driven analyses are applicable.

Record_job_launch(Job_entities, Node_list, Time, DB, Status)

IN	Job_entities	A set of information associated with the job (job ID, user ID, binary, etc.)
IN	Node_list	List of node IDs the job is launched
IN	Time	Time stamp of the job launch
IN	DB	ID to designate the database
IN	Status	List of errors observed during the procedure

Record_job_retire(Job_entities, Node_list, Time, DB, Status)

IN	Job_entities	A set of information associated with the job (job ID, user ID,

		binary, etc.)
IN	Node_list	List of node IDs the job was running
IN	Time	Time stamp of the job retirement
IN	DB	ID to designate the database
IN	Status	List of errors observed during the procedure

The following function is utilized in the Signature Handler inside the Monitor tool. It searches and fetches the most relevant profile to the target job from the database.

Get_profile(Job_entities, DB, Profile, Status)

IN	Job_entities	A set of information associated with the job (job ID, user ID, binary, etc.)
IN	DB	ID to designate the database
OUT	Profile	Profile data of a previous run of this job/application (Null if no relevant profile is found)
OUT	Status	List of errors observed during the procedure

Then, the following function generates the signature to characterize the job, which is based on the profile (Profile) obtained with the above function or is calculated using the statistic acquired at runtime or simply only with the job entities, which depends on the implementation or the availability of the profile. The interface is used by the Estimator.

Genarate_signature(Job_entities, Dat_src, Signature, Status)

IN	Job_entities	A set of information associated with the job (job ID, user ID, binary, etc.)
IN	Dat_src	A variable to specify whether use the profiled data or collected data at runtime
OUT	Signature	Signature to characterize the job (e.g., a list of parameters to characterize the job such as compute/memory intensity)
OUT	Status	List of errors observed during the procedure

The following functions are provided by the Monitor to other components, and internally the Estimator predicts power/performance/energy of the target by using the profile or runtime monitoring data specified by Dat_src.

Get_pow_estimation(Job_entities, Res_alloc, Dat_src, Est_pow, Status)

IN	Job_entities	A set of information associated with the job (job ID, user ID, binary, etc.)
IN	Res_alloc	A set of information to state resource allocations (# of nodes, power cap, etc.)
IN	Dat_src	A variable to specify whether use the profiled data (DB) or collected data at runtime
OUT	Est_pow	Estimated power consumption
OUT	Status	List of errors observed during the procedure

Get_perf_estimation(Job_entities, Res_alloc, Source, Est_perf, Status)

IN	Job_entities	A set of information associated with the job (job ID, user ID, binary, etc.)
IN	Res_alloc	A set of information to state resource allocations (# of nodes,

		power cap, etc.)
IN	Dat_src	A variable to specify whether use the profiled data (DB) or collected data at runtime
OUT	Est_perf	Estimated performance for the job
OUT	Status	List of errors observed during the procedure

Get_ene_estimation(Job_entities, Res_alloc, Source, Est_ene, Status)

IN	Job_entities	A set of information associated with the job (job ID, user ID, binary, etc.)
IN	Res_alloc	A set of information to state resource allocations (# of nodes, power cap, etc.)
IN	Dat_src	A variable to specify whether use the profiled data (DB) or collected data at runtime
OUT	Est_energy	Estimated energy for the job
OUT	Status	List of errors observed during the procedure

5.3. Interfaces and Use Cases

In this section, we describe how the interface/module functions can be applied to support the implementation of several use cases.

[Basic]
The site administrator firstly checks the range of the system power cap using the function *Get_sys_pow_cap_range()* and then sets it up within the range by using *Set_sys_pow_cap()*. She/he can also check the current system power cap by using *Get_sys_pow_cap()*. Once the SPM receives the power cap change, it internally decides the node-level power cap (set evenly across nodes) and then sends out the node power cap value by using *Send_node_pow_cap()*. Then the node manager receives the power cap by the function *Recv_node_pow_cap()* that snoops the change, and then updates the node power cap. It returns the acknowledgement to tell if it completes properly or not. In case of an error, the node manager sends the error to the SPM, and the SPM then notifies the error to the site administrator. To conduct these procedures the following interface functions are also used: *Set_node_pow_cap()*; *Get_node_pow_cap()*; *Get_node_pow_cap_range()*.

[Basic+]
In addition to the entire system power capping, this use case sets up the thermal capping to all the nodes evenly. The entire procedure is almost same as the power capping except for controlling temperature, and the following functions are additionally needed: *Get_sys_temp_cap_range(); Set_sys_temp_cap(); Get_sys_temp_cap()*; *Send_node_temp_cap(); Recv_node_temp_cap(); Set_node_temp_cap()*; *Get_node_temp_cap()*; and *Get_node_temp_cap_range()*.

[Basic++]
This use case provides the anomaly detection functionalities. To this end, it needs to continuously monitor the system state and notify an error/anomaly if detected. To this end, it can use the monitoring functions listed in 5.1.1 and check if the power/thermal capping is working properly. Further, it can use a more sophisticated anomaly detection mechanism

using the runtime monitoring data or the historical data recorded on the database. The estimation function is in practice implemented as a plugin for the monitor or other tools.

[SysThru/SysEne]
In addition to the basic functionalities described above, this use case applies an optimization on node power cap allocations depending on the characteristics of running jobs (e.g., compute or memory intensity). We can consider two different approaches here: (1) profile-driven static and proactive allocations; and (2) dynamic and reactive allocations based on runtime measurement. For the former, we utilize the profile-related functions: *Send_job_entities()*; *Recv_job_entities()*; *Record_job_launch()*; *Record_job_retire()*; *Genarate_signature()*; *Get_profile()*; *Get_pow_estimation()*; *Get_perf_estimation()*; *Get_ene_estimation()*. One or more estimation functions are selected from the last three depending on the objective of the optimization. In addition, to realize the dynamic and reactive power cap optimization, *Request_pow_budget()* and *Return_pow_budget()* are used.

[AppPerf/AppEtS]
These two use cases allow application developers to tune the trade-off between power and performance. For instance, the clock frequency can be scaled down with a negligible performance overhead while waiting for the completion of data transfer across nodes, which can be identified by analyzing the code. To this end, the power/performance setup function, such as *Set_mod_pow_cap()*, *Set_mod_freq()*, and others can be exposed to the developers while the node manager needs to check if the modification does not violate the node-level power/thermal capping. Further, they may utilize the profiles of previous runs provided by the Monitor for the fine tuning purpose.

[NodPowShift]
This use case distributes the power budgets to hardware modules inside a node in order to maximize/minimize a given objective while keeping the node-level power cap set by the SPM. To this end, it needs per-module power control interfaces listed above. The exact control algorithm/mechanism is in practice given by a plugin in the node manager.

[SchedOpt]
This use case co-optimizes the job scheduling and the power budgeting across nodes. To this end, the RJMS interacts with the SPM to obtain the current system power management status – the most prominent one is the remaining power budget given by *Get_remain_pow_budget()* – and then decides the job launch decisions based on the profile-based power, performance, or energy estimations provided by *Get_pow_estimation()*, *Get_perf_estimation()*, or *Get_ene_estimation()*, respectively. The scheduling algorithm can also be trained by using the historical data stored on the database, using such as reinforcement learning.

5.4. Discussions

The above architecture and interface descriptions are still in the intermediate status based on the use cases and their requirements defined in WP1 as well as the current status of implementations/integrations ongoing in WP3, and thus can be continuously elaborated from both top-down and bottom-up directions. Further, in the final version, the descriptions will be more comprehensive by covering sophisticated use cases investigated in WP2 and workflow

engine paths along with power management. To prove the completeness of the set of functions, we will provide pseudo codes to realize the use cases while using the functions in the final version. In WP3, a set of the use cases will be implemented in the same way as the pseudo codes. Therefore, the real-world evaluations using the integrated software will be a proof of the correctness of the architecture design here.

There are several missing pieces not included here but mentioned in the previous deliverable D1.1. One of them is the token accountant that calculates the token consumption, i.e., how much the site charges for executing a job under power management. In most HPC sites, the token consumption is usually determined by the number of nodes occupied by the job, multiplied by the job execution time. In case of application-level power or energy optimizations (e.g., the AppEtS use case), the site admin should motivate the users to be green, otherwise most of them would optimize their application to just minimize the time to solution, even if the job-oriented power/energy optimization use cases are applied. Revisiting the token management policy is one of the promising solutions for this purpose – if the token consumption were directly determined by the job power or energy consumption, the users would care about it. Even though this function is out of scope for our integrations in WP3, covering the token accounting aspects at the architecture level in WP1 is a promising future option.

6. PowerStack Integration

In this section, we first introduce an initial assessment of our software tools to realize the PowerStack path, i.e., we map them to the architecture components specified in the last section and discuss the missing pieces in order to support these use cases. We then move on to the initial integration plan based on this assessment. We then finally introduce the summary of our intermediate integration status ongoing in WP3, i.e., how these use cases are converted into several integration instances.

6.1 Tool Assessment for PowerStack

TABLE 9: Candidates for System Manager / Monitor and Current Functionality Support / = Fully Supported or Need Minor Updates, X = w/ Some Restrictions or Need Moderate/Major Updates, Blank = No (or Almost No) Support

	System Manager			Monitor						
	RJMS	SPM	API/Lib	Sensors		Estimator		DB	Signature	API/Lib
				In band	Out of band	Anomaly	Pow/perf			
SLURM /OAR	/	X	/	/		X		/		/
DCDB			/	/	/	X	/	/	/	/
Examon			/	/	/	X	/	/	/	/
BEO				/	X	X				/
EAR		X	/	/		X		/		/

TABLE 9 lists the candidate tools for System Manager and Monitor, and the support for the key functionalities in the early stage of the project. SLURM and OAR are RJMS tools, and they support key functionalities including job scheduling and token management. They also support other functionalities with respect to power management and statistics recording, however there are some other tools more relevant to these roles. To implement the SchedOpt use case, a new scheduling plugin needs to be developed. As for the SPM, EAR (EARGM) is one of the best options because the interaction with SLURM to obtain job information is already supported in the tool via SLURM plugins/APIs. As for the statistics analysis functions, several monitoring tools (e.g., DCDB and Examon) already support them, such as ML-based modelings, and they are extensible and changeable by plugins. These analytics functions can be extensible and will be useful for a variety of use cases.
As for the monitoring aspect, DCDB and Examon can measure both in-band and out-of-band sensors periodically (from 0.1Hz up to 100Hz of frequency depending on what we measure) and are extensible to support any sensors by plugins, including such as those in cooling facilities. These collected information is recorded with the associated job information on their database – these tools are also able to interact with SLURM to obtain job information. The measurement function is accessible by other tools, such as those Node Manager tools, by

using their APIs. For the above strengths, DCDB and Examon are selected for the mainstream option of the Monitor module, however the others also can work as the Monitor in several implementations depending on the use cases.

TABLE 10: Candidates for Node/Job Manager and Current Functionality Support / = Fully Supported or Need Minor Updates, X = w/ Some Restrictions or Need Moderate/Major Updates, Blank = No (or Almost No) Support

	Node Manager			Job Manager			HW access	
	Closed loop ctr	Access to sys mngr	API/Lib	Profiling	Pow control	API/Lib	In-band	Out-of-band
SLURM	X	/	/				/	
PULP ctr	/						/	X
BEO	X	X	/				/	X
BDPO				/	X	/	/	
EAR		/	/	/	X	/	/	
Countdown					X	/	/	

Next, TABLE 10 summarizes the candidates for Node or Job Managers, and the current support for their key functionalities as well as their in-band or out-of-band hardware knob/sensor accessibilities. The SLURM node daemon works together with the SLURM system controller, and there are a variety of API functions to access their information. Although the default power management support in the node daemon is limited (e.g., no node-level power shifting support), the tool is useful as an interface to interact with the system manager. PULP controller is a low-level power controller, works transparently to the application, user, and system software, currently targeting EPI processors [6]. The tool can access both in-band and out-of-band sensors and automatically optimizes power management knobs using model predictive control algorithms under thermal and power limits. BEO is an out-of-band power monitoring and controlling tool. The supported hardware is a set of AMD/Intel machines in the Atos catalog because the tool is developed in Atos, with a particular focus on their products, and a plugin is needed for other systems. It monitors power consumption using out-of-band sensors and can set the power cap using the in-band RAPL interface. The tool is going to be improved by implementing the following: setting the node power/termal capping via Slurm (Basic/Basic+); and sophisticated power control and anomaly detection mechanisms. BDPO is a job-oriented profile-based power-performance optimization tool, which optimizes clock frequency to trade-off performance and energy or to minimize energy (AppEtS use case supported), and can be extensible to cover other job-oriented use cases. EAR is another job-oriented power-performance monitoring tool. It transparently optimizes the power management knobs on CPUs and GPUs using the profiles of previous runs that are automatically detected. The power management policies are implemented as plugins. The tool currently supports the minimizing energy to solution (AppEtS) use case, and is extensible to cover other use cases such as AppPerf or NodPowShft. COUNTDOWN is another tool that enables job-level power/performance optimizations based on a different focus than others. It tries to minimize the power consumption while waiting for the completion of an MPI communication, by scaling down the clock frequency or going into one of the CPU sleep states (C-state). It targets Intel CPUs, but

is going to support other hardware including GPUs. The tool will reinforce the AppEtS use case and will open up new research opportunities and use cases, which are going to be covered in the future deliverables.

6.2 Summary of current integration status

Based on the above tool integration assessment, we determined several integration scenarios (or instances). Figure 7 depicts the general concept with respect to how we convert these use cases into the integration scenarios. Each integration scenario supports its own use cases depending on what functionalities their tools support. In general, the site administrator can pick one use case suitable for the objective, which is in practice realized by selecting the associated configuration and plugin. The implementation/integration is conducted in the WP3, and the more detailed descriptions are provided in the deliverable D3.1. In addition to this, the WP2 extends/sophisticates the use cases and actual integrations.

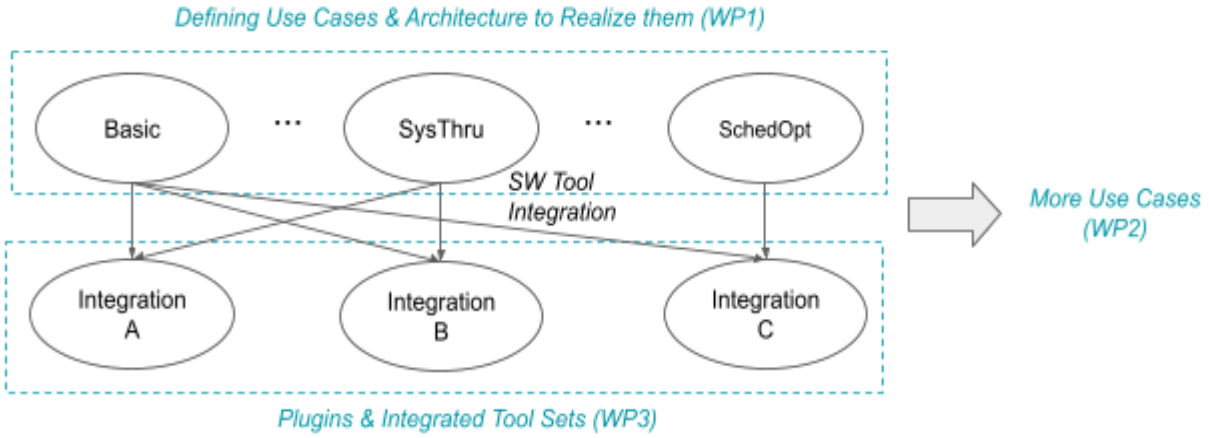

Figure 7: Mapping of Use Cases into Integrations

Integration Scenario #1: Application-aware system power capping

This integration scenario aims at maximizing total system throughput under a power cap. The system throughput is maximized thanks to the feedback that is manually provided by end-users to the Job scheduler about the performance behavior of the job or the one that is automatically computed by the Execution Profile Compute Module (EPCM), which works as a signature handler. The feedback is then conveyed to the SPM which generates per-job power limits for each node. In this scenario, the SPM plays the key role in terms of global power management at the system scale, and in terms of power capping strategies at the job scale.

The relevant use cases and the involved software tools are as follows. As described above, this scenario is relevant to SysThru, in particular profile-driven proactive power management. This also covers a more naive Basic use case as it simply distributes the power cap evenly across nodes.

Relevant use cases	Basic, SysThru

Involved software tools	OAR(RJMS), BEO(SPM), ECPM(Sig Handler), EXAMON/DCDB(Monitor)

To realize this integration scenario, we integrate OAR, BEO, EXAMON/DCDB, and ECPM. The ECPM is a newly developed external module to enable the signature computation functionality.

Integration Scenario #2: Application-aware energy optimization under a system power cap

This integration scenario is complementary to the previous one. As in the previous scenario, the Job scheduler and the Node manager enforce the power cap. In addition, the Job manager and the Node manager optimize the power state of the compute resources based on application demand. The Monitor provides insights to the system administrator on platform metrics and to the users on job's efficiency through dashboards.

The relevant use cases and the involved software tools are as follows. This scenario covers application-level power optimizations (AppThru, AppEne), while following the power constraint set by the system and node manager.

Relevant use cases	AppThru, AppEne
Involved software tools	BEO(SPM), EAR/BEO(Node Manager), EXAMON(Monitor), COUNTDOWN(Job Manager)

To realize this scenario, BEO, EAR, EXAMON, and COUNTDOWN are integrated. In particular, the COUNTDOWN plays an important role in this scenario by setting the compute resources into a low power mode while waiting for MPI communications.

Integration Scenario #3: Application-aware power capping with job scheduler support

This integration scenario covers a use case where the RJMS plays an active role in the power management. In this scenario, the SPM applies the cluster powercap algorithm to dynamically setting the node powercap based on application characteristics. The RJMS receives information from the SPM about the system status in terms of power consumption. Based on this information, the RJMS can take several actions/decisions: It can (1) influence the scheduling policy, the order of jobs, and job priorities in order to adapt the scheduling to the system status, and (2) it can force the System Power Manager to reduce the allocated power of running nodes in order to guarantee some power for new jobs. The following table lists the relevant use cases and the involved software tools. In addition to SysThru, NodPowShift is also applicable to this integration scenario thanks to EAR's multiple device support.

Relevant use cases	Basic, SysThru, SchedOpt, NodPowShft

Involved software tools	OAR(RJMS), EARGM(SPM), EARD(Node Manager), EAR/Examon/DCDB(Monitor), EARL/COUNTDOWN(Job Manager)

7. Evaluation plan

7.1 REGALE Strategic objectives and KPIs

The evaluation plan for REGALE is aligned to the Strategic Objectives of the project as described in the DoA and summarized in Table 11 together with the relevant KPIs and targets. The objectives of the project are both quantitative and qualitative. SO1 together with its subobjectives 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 solution and SO3 is relevant to widening the use of supercomputing resources to more complex, next generation applications. This document will provide a detailed plan on how to evaluate the tasks within REGALE that have quantitative KPIs and will also provide initial guidelines on how to assess the qualitative targets.

Table 11: Summary of REGALE Strategic Objectives and the relevant KPIs

Strategic Objective		**Baseline**	**KPI**	**Target**
SO1	SO1.1: Improved application performance.	Execution in SoTA HPC systems	Quantitative: FLOPS or time to solution	20%
		Pilots before REGALE		2x
	SO1.2: Increased system throughput.	SoTA HPC systems	Quantitative: jobs / hour	30%
	SO1.3: Minimization of performance degradation under the operation with power constraints.	SoTA HPC systems	Quantitative: Decrease throughput penalty	50% less penalty
	SO1.4: Decreased energy to solution.	SoTA HPC systems	Quantitative: energy reduction	10%
SO2	Scalability	n/a	Quantitative	exascale
	Platform independence	n/a	Qualitative	n/a
	Extensibility	n/a	Qualitative	n/a
SO3	Automatic allocation of resources	Traditional HPC application development / deployment	Qualitative	n/a
	Programmability	Traditional HPC application development		n/a
	Flexibility	Traditional HPC application deployment	Qualitative	n/a

7.2 Evaluation targets

REGALE includes a number of targets (or tasks) that require quantitative evaluation, these being the REGALE prototypes, the five pilots and sophistication tasks from WP2. Each of these targets is analyzed in the next sections.

7.2.1 REGALE prototypes

The prototypes of REGALE (see D3.1) 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. 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: a) an overview of the current state of the art in energy/power-aware supercomputing systems, b) the value proposition of REGALE and c) the proposed evaluation scenario that will compare the relevant state-of-the-art setups with REGALE's prototypes in order to assess whether the strategic objectives are met.

State of the art: When setting the goal of exascale computing almost a decade ago, it was realized that energy consumption and excessive power needs would be the most critical obstacles to achieving this goal. Technological enhancements have been applied on several components of the supercomputing ecosystem, most notably at the datacenter and hardware levels. Regarding supercomputing installations, hot water and free cooling technologies were promoted [8, 9, 10]. On the hardware side, higher integration densities with more vector and SIMD units and smarter power controls have already been integrated on modern datacenter processors [11, 12]. These aforementioned techniques are orthogonal to REGALE's objectives, with an important note that our approach heavily relies on the power management capabilities provided by modern processor technology.

We have now entered a phase where energy efficiency is required not only at the design and implementation of a supercomputing data center, but also during its operation. A number of research works has demonstrated the benefits of sophisticated power management [13, 14, 15, 16, 17] and tools to monitor and control power at various levels (node, rack, system) of the site have been implemented and deployed. The PowerAPI specification defines a set of abstractions and interfaces between various actors and HPC system components for power/energy measurement and control, easing the implementation of a software stack for power management on multiple architectures. The HPC PowerStack working group takes this effort a step further, fostering the standardization of the power/energy management software stack for HPC systems, by generalizing node-level hardware/software interfaces.

Based on the aforementioned trend, a number of supercomputing sites have enhanced their software stack with power monitoring and management tools. EAR (https://gitlab.bsc.es/ear_team/ear) is the system software for energy management tools used in SuperMUG-NG (LRZ) since September 2019. SuperMUC-NG (https://doku.lrz.de/display/PUBLIC/SuperMUC-NG) is a homogeneous system composed of 6480 computational nodes with Intel CPUs. In SuperMUC-NG energy optimization is applied by default to all the jobs executed. The optimization policy selected is min_time_to_solution with a default frequency of 2.3GHz (the nominal is 2.7GHz). Jobs executed without energy optimization are executed at a fixed CPU frequency of 2.3GHz. SuperMUC-NG uses an EAR cluster powercap strategy called Soft-powercap. This strategy modifies the node powercap (setting it on or off) depending on the total power consumption. By default, the node

powercap is off but it is enabled in case the total power consumption reaches a certain value. The node powercap is disabled if the total power consumption is below a lower limit. Energy accounting is reported for all the jobs and node and cluster monitoring are also implemented using EAR services. More details can be found here (https://doku.lrz.de/display/PUBLIC/Energy+Aware+Runtime).

Quite recently (2021) EAR was also installed in Snellius, the National Dutch supercomputing center in SURF (https://www.surf.nl/en/dutch-national-supercomputer-snellius). Snellius is a heterogeneous system with two main partitions, one with AMD Zen2 CPUs and a second one with Intel CPUs + NVIDIA GPUs. In Snellius EAR takes care of the system monitoring and job accounting. The EAR library with monitoring or optimization policies has also been enabled but is not forced to be used by default. According to the current plans. EAR will also be installed in the next MareNostrum5 system in the Barcelona Supercomputing Center.

As a summary, the current state-of-practice in supercomputing operation includes systems that a) are completely energy/power-unaware and perform brute-force power capping 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 **there exists no supercomputing center that performs some kind of sophisticated energy management or power capping**.

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%.

Evaluation scenario: The evaluation scenario in this case involves the comparison of the achieved throughput (jobs/time unit) in a state-of-the art supercomputer setup with a similar setup with the REGALE prototype activated under the application of several levels of power capping. Our goal is to reach the highest possible number of nodes (minimum target: 256 nodes) with the important note that the REGALE prototype requires privileged access, a fact that limits access to large-scale production systems. In any case, we will maximize efforts to provide convincing large-scale results either by actual medium/large-scale measurements or by a combination of these measurements with modeling and simulation results.

7.2.2 REGALE pilots

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, this optimization problem is carried out using evolutionary algorithm (EA) tools with some noticeable add-ons. The latter include: a) surrogate evaluation models (metamodel-assisted EA; MAEA) to avoid "useless" calls to the expensive CFD evaluation software 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. Thus, we get real optimal designs (in accordance with the selected blade shape parameterization and the imposed bounds). Over and above, the flow solution and, as a consequence, the optimization run become faster due to the use of a GPU-accelerated CFD software.

Irrespective of the shape optimization itself, the computed performance of hydraulic machines is certainly affected by the imposed boundary conditions and "feel" uncertainties in the flow rate, the inlet flow angle etc. The obtained optimal solutions must correspond to robust designs, which means that they should slightly be affected by any uncertainty in the boundary conditions etc. This is measured by an Uncertainty Quantification (UQ) technique. A nice way to perform UQ is by using the non-intrusive variant of the Polynomial Chaos Expansion (niPCE) method. The PCE requires a series of flow computations in various geometries (shapes), 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 can be performed through the Melissa workflow manager using a storage-free approach and with the ‘optimal’ allocation of computational resources for concurrent simulations.

REGALE value proposition: All optimization runs were/are performed on a computational node with 4 Nvidia A100 GPUs. Comparisons between the standard EA and the PCA-driven MAEA in terms of the optimization turnaround time and/or the obtained optimized solution(s) for a given computational budget were made.

Regarding UQ studies, the same computational node is used and comparisons on the required time and storage to compute the statistical moments with and without the use of Melissa will be carried out.

Evaluation scenario: The above mentioned 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.

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 thousand) 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, store the results to disk, often called ensemble data, to later read them back from disk to compute the required statistics. The amount of storage needed may quickly become overwhelming, with the associated long read time that makes statistical computing 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 will collaborate to run a petabyte-level multiple simulations study of an industrial case. EDF will provide the model and parameters for large-scale CFD simulations using Code_Saturne (https://www.code-saturne.org/).

Evaluation scenario: The simulations will run as a large ensemble and the produced data will be 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.

Pilot 3: Enterprise Risk Assessment

State-of-practice: Through the OLISTIC Enterprise Risk Management Platform, we provide risk assessment for organisations. This is achieved through a representation model of assets, their vulnerabilities, and a graph that depicts all applicable asset interconnections. This assessment is also enriched by collecting and analysing network metadata for detecting possible issues and utilizing security information from different online sources. Such features are very demanding in terms of data size and processing capabilities needed. To detect and calculate the risk of Advanced Persistent Threats (APTs), analysis on 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. For example, the operation of only one 24-port gigabit router may generate, under full utilization, approximately 50TB of data on a daily basis. Also, finding all available paths in a graph can have exponential time complexity, which makes calculations even more difficult as the topology increases. Hence, high-performance computing can significantly decrease the time needed to identify threats and to solve the attack paths enumeration problem, in comparison to existing server setups that cannot easily achieve this kind of processing.

REGALE value proposition: Utilization of a 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 accordingly 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 in the context of REGALE (using the RYAX Platform) is the reduction of the response time of the Threat Analytic Models, enabling the accurate detection of threats. Moreover, a dataset generated containing network metadata of our servers is used to evaluate the consumption via messaging brokers(Kafka), the Spark modules responsible for capturing data in a time-window fashion and detection of network spikes. Such disturbances are forwarded in the alerting mechanism via OLISTIC. By comparing the workflow deployed by RYAX in an HPC environment to the original, cloud based approach, we can gain insights into the speedups and the improvements of HPC acceleration provided by REGALE.

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 customising 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
- Optimise 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.

Pilot 5: Design of car bumper made of carbon nanotube reinforced polymers Scope: Improve performance of stochastic multiscale reinforced polymers

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. In terms of programmability, the REGALE prototype architecture facilitates the integration of various modules (e.g. monitors, node managers, job managers) with our in-house code for an efficient deployment on supercomputers. The REGALE tools will ensure the optimal resource allocation for our application, while maximizing throughput. In addition, 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 scenario: To assess the benefits offered by the REGALE project, we will focus on two scenarios. For the first scenario, the baseline will be the time to completion achieved in the execution on a single machine (6xIntel i7x980 CPUs with hyperthreading and an NVIDIA GTX 580) and we will compare against the time to completion in a supercomputer, where the REGALE tools are installed. For the second scenario, we will compare the time to

completion and the storage requirements for the pilot with and without the integration of the Melissa workflow manager. The KPI in both cases will be the application speedup.

7.2.3 REGALE sophistication

Table 12 summarizes the evaluation plans for the sophistication efforts of WP2.

Table 12: Evaluation plan for WP2 tasks

Task	SoTA / baseline	Value proposition	KPI	Target
Multi-node co-scheduling	SoTA resource managers with allocation at node granularity	Improved throughput with co-scheduling (allocation at “half-CPU” granularity)	System throughput (jobs/sec)	>15%
GPU co-scheduling	Allocation at GPU granularity	Resource partitioning on heterogeneous/ homogeneous node under power cap	GPU throughput (jobs/sec) under power cap	>30%
Power-aware scheduling	SoTA batch schedulers	Power aware scheduling in OAR	System throughput (jobs/sec) under power cap	>5%
Power/thermal control	CPU power and thermal controller of major vendors. Intel, ARM SCP, IBM OCC	Combined thermal and power control with application performance pinning	Application performance under node/cpu power and thermal cap	>5%
Application-aware power capping	Intel CPU-centric nodes or GPU-centric nodes with in-band power-capping capabilities	Application-aware power-capping on a partition of nodes	System throughput (jobs/sec)	>15%
Elasticity for Big Data applications	Big Data applications executed on HPC	Elasticity for Big Data/ streaming applications on HPC systems	Turnaround time of Big Data/Spark jobs	20%

7.3 Available platforms and timeline

Up to the time of the writing of this document, the partners have secured access to the platforms presented in Table 13. Note that this is not an exhaustive list, it includes systems available for all REGALE partners. Based on their own facilities or collaborations, partners may utilize access to other platforms for specific evaluation tasks.

Table 13: Availability of platforms for REGALE evaluation

Name	Node architecture	Number of nodes	Access mode	Appropriate for
ICCS	2x x86 Ice lake	6	root	Development platform (e.g. Pilots 3 ,4)
GRID 5000	2x x86	128	root time restrictions apply	REGALE prototype Co-scheduling Power-aware scheduling
Armida	ARM + V100 GPU	8	user	Development platform (REGALE prototype)
E4 - Intel	2x Xeon Gold	4	root	Development platform (REGALE prototype)
CoolMUC-2	2x Intel Haswell	812	user	Pilot 2
G100	2 x CascadeLake	422	user (with access to some power knobs)	REGALE prototype Pilot 2
EuroHPC#1	1x Xeon 4x A100	3456	user	Pilot 1, pilot 5
EuroHPC#1	2x Intel Sapphire	1536	user	Pilot 2

The evaluation plan for REGALE will follow the project's timing. In particular, we will proceed as follows:
Q2 2023: Testbed access and software setup
Q3 2023: Initial testing experiments
Q4 2023: Initial results and feedback to development and integration tasks
Q1 2024: Collection of final evaluation results

7.4 Evaluation of qualitative objectives

This deliverable presented an evaluation plan 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 intend to validate the success of the qualitative objectives and refer to the deliverables that more information is or will be provided.

SO2: Platform independence. Platform independence will be validated through the integration of alternative integration scenarios involving different hardware architectures and components. The REGALE API definition is also expected towards this direction. More information will be provided in Deliverable D3.3.
SO2: Extensibility. Extensibility is sought by the ease incorporation of new features and alternative modules. This will be achieved by the REGALE API definition and instantiation. More information will be provided in Deliverables D3.3 and D5.10.

SO3: Automatic allocation of resources. Atomic allocation or resources refers to the integration of the five REGALE pilots with the relevant workflow engines. The results of this integration are reported in D4.1, D4.2 and will be finalized in D4.3.
SO3: Programmability. This will be qualitatively assessed by the application developers and pilot users of the Consortium by comparing the features of their application before and after the optimizations within REGALE. More details will be provided in D4.3.
SO3: Flexibility. This objective will be validated by the ability to execute under lightweight virtualization within the REGALE-enabled system.

8. Conclusions and Future Directions

In this intermediate deliverable, we sorted out the architecture requirements, the components/interfaces definition, the mapping to our integrations, and the evaluation plan. To this end, we first collected the current status of our software tools and formulated possible PowerStack use cases in a general and incremental manner. Second, we specified the software requirements for each use case and described the open REGALE architecture with interfaces, while following the requirements. We then assessed our software tools to realize the use cases based on the architecture and introduced the current status of our integrations and evaluation plans. This deliverable is going to be the milestone for any software implementations to realize power management in HPC systems as well as the pointer to missing pieces investigated as scientific studies in WP2.

In the final deliverable, we will update the REGALE architecture in accordance with the lessons learned throughout implementing the use cases in WP3. We assume the integration experiences would be useful to elaborate the architecture definition. However, this would require minor modifications and would not change the use case definition, though, because the use cases and the requirements are the guideline for the actual implementations and should be independent of them. Improving the architecture descriptions by using an open/standard form by following the literature [7] will be another good option.

Another direction is generalizing the architecture one step further by unifying the PowerStack path and the workflow engine paths (Melissa and RYAX). In WP4, our pilot applications will be integrated with these workflow engines to realize sophisticated simulations such as automatic parameter sensitivity analyses, simulation surrogates with deep neural networks, automatic concurrency controls, and so forth. These new paradigms of application management in HPC would introduce new research opportunities to holistic resource management in HPC centers, in particular when under a power constraint. For instance, the system manager might need to be aware of the behaviors of those new types of scientific simulations for better power budgeting across different jobs, nodes, or components. Another promising way as an example is implementing a power capping functionality inside of these workflow engines and coordinating the system manager and the workflow engines to deal with the power budgets in a sophisticated way via a newly introduced or an extended interface between them. This could be a new use case, and then we would update the requirement specifications as well as the architecture accordingly. This work would need an extensive collaboration across different paths.

Aside from the workflow engine aspects, we will introduce any other research outcomes from WP2 to the use case specifications and the architecture design. In WP2, we will generally

investigate various sophisticated resource management techniques to enhance the state-of-the-arts, all of which would bring us useful insights to elaborate our architecture. As an example, co-scheduling, i.e., co-locating multiple HPC jobs on a node in a space-sharing manner would be a great addition. We will explore this direction in terms of both the theoretical aspects including hardware/software requirements and the actual implementations. For instance, clarifying the requirements to partition the power budgets among co-located jobs on the same node at the same time would be a good option for fair power/energy accounting. As another example, we are investigating ML-based resource management techniques, which would help with specifying the roles or requirements for some sub-components in these architecture modules in more detail. Furthermore, covering different kinds of nodes or facilities in our power management loop is another promising direction to extend our work. As described in this document, including I/O nodes is one good option for this because various applications are now I/O bound, and thus dealing with power budgets across compute and I/O nodes in both system- and application-level would be a good use case to consider. Covering cooling systems in our power management loop is a promising direction to explore, though we need a software layer to power cap them, predict their power consumption, detect anomalies and need to care about the time constant to converge to the target value. More generally, power budgeting across different kinds of compute nodes for inter-node heterogeneous HPC systems would be another option to explore. Furthermore, investigating how we should handle anomaly states at different granularities including different kinds of hardware would be an interesting research direction.

Finally, in this deliverable we provided an evaluation plan that will guide the evaluation tasks of REGALE. In particular, we described the baseline for each one of the project targets that needs to be quantitatively evaluated, set the relevant KPIs and targets and sketched the evaluation process that needs to be followed in order to assess the success of each of the targets. A number of evaluation platforms have already been identified and need to be further refined in the upcoming period of the project.

References

[1] “The HPC PowerStack.” *The HPC PowerStack | HPC PowerStack Seminar Website*, https://hpcpowerstack.github.io/. Accessed 28 November 2021.
[2] “TOP500.” *TOP500*, https://www.top500.org/lists/top500/2021/11/. Accessed 28 November 2021.
[3] Pandruvada, Srinivas. “Running Average Power Limit – RAPL | 01.org.” *Intel Open Source Technology Center*, 6 June 2014, https://01.org/blogs/2014/running-average-power-limit-%E2%80%93-rapl. Accessed 30 November 2021.
[4] “NVIDIA System Management Interface.” *NVIDIA Developer*, https://developer.nvidia.com/nvidia-system-management-interface. Accessed 30 November 2021.
[5] Dey, Somdip, et al. "EdgeCoolingMode: An agent based thermal management mechanism for dvfs enabled heterogeneous mpsocs." *2019 32nd International Conference on VLSI Design and 2019 18th International Conference on Embedded Systems (VLSID)*. IEEE, 2019.
[6] “EPI: European Processor Initiative” EPI, https://www.european-processor-initiative.eu/. Accessed 9 December 2021.
[7] Maiterth, Matthias. *A Reference Model for Integrated Energy and Power Management of HPC Systems*. Ludwig-Maximilians-Universität München, 2021.
[8] H. Shoukourian, et al. Analysis of the efficiency characteristics of the first high-temperature direct liquid-cooled petascale supercomputer and its cooling infrastructure, JPDC, vol. 107, pp. 87– 100, 2017
[9] C. Conficoni, et al. Integrated energy-aware management of supercomputer hybrid cooling systems, IEEE Transactions on Industrial Informatics,vol. 12, no. 4, pp. 1299–1311, Aug 2016.
[10] C. Conficoni, et al. Hpc cooling: A flexible modelling tool for effective design and management, IEEE Transactions on Sustainable Computing, pp. 1–1, 2018.
[11] J. Dongarra, et al. Top500 supercomputer sites, https://www.top500.org/lists, 2019, accessed 29 March 2019.
[12] W.-c. Feng and K. Cameron, The green500 list: Encouraging sustainable supercomputing, vol. 40, no. 12. IEEE, 2007.
[13] F. Fraternali, et al. Quantifying the impact of variability and heterogeneity on the energy efficiency for a next-generation ultra-green supercomputer, IEEE TPDS, vol. 29, no. 7, pp.1575–1588, 2018.
[14] A. Auweter, et al. A case study of energy aware scheduling on supermuc, in Supercomputing, J. M. Kunkel,T. Ludwig, and H. W. Meuer, Eds.Cham: Springer, 2014, pp. 394–409.
[15] C. Hsu and W. Feng, A power-aware run-time system for high-performance computing, SC ’05.
[16] A. Borghesi, et al. Scheduling-based power capping in high performance computing systems, Sustainable Computing: Informatics and Systems, vol. 19, pp. 1–13, 2018.
[17] S. Bhalachandra, et al. An adaptive core-specific runtime for energy efficiency, IPDPS’17, pp. 947–956.