H2020-JTI-EuroHPC-2019-1

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

Grant Agreement Number: 956560

D1.1

REGALE Requirements, Initial Architecture, and Evaluation Plan

Final

Version: 1.0

Author(s): Eishi Arima (TUM), Bruno Raffin (UGA), Yiannis Georgiou (RYAX), Daniele Cesarini (CINECA), Nikela Papadopoulou(ICCS), Mohsen Seyedkazemi Ardebili (UNIBO), Abdelhafid Mazouz (ATOS), Millian Poquet (UGA), Andrea Bartolini (UNIBO), Pierre-François Dutot (UGA), Martin Schulz (TUM), Georgios Goumas (ICCS)

Contributor(s): Listed in Acknowledgement section

Date: 21.12.2021

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: D1.1</td></tr>
<tr><td colspan="2">Deliverable Nature: Report</td></tr>
<tr><td rowspan="2">Dissemination Level: PU *</td><td>Contractual Date of Delivery:
31 / 12 / 2021</td></tr>
<tr><td>Actual Date of Delivery:
21 / 12 / 2021</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: D1.1</td></tr>
<tr><td>Version: 1.0</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_D1.1_Architecture_ver1.0.pdf</td></tr>
<tr><td rowspan="3">Authorship</td><td>Written by:</td><td>Eishi Arima (TUM), Bruno Raffin (UGA), Yiannis Georgiou (RYAX), Daniele Cesarini (CINECA), Nikela Papadopoulou(ICCS), Mohsen Seyedkazemi Ardebili (UNIBO),Abdelhafid Mazouz (ATOS), Millian Poquet (UGA), Georgios Goumas (ICCS)</td></tr>
<tr><td>Contributors:</td><td>Listed in Acknowledgement section</td></tr>
<tr><td>Reviewed by:</td><td>Andrea Bartolini (UNIBO), Pierre-François Dutot (UGA), Martin Schulz (TUM)</td></tr>
</table>

	Approved by:	Georgios Goumas (ICCS), Martin Schulz (TUM)

Document Status Sheet

Version	Date	Status	Comments
0.1	01.12.2021	Draft	Initial version
0.2	09.12.2021	Draft	Second version
0.3	17.12.2021	Draft	Internal review
1.0	21.12.2021	Final	Minor Changes

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 initial software architecture (Task 1.3), and the initial evaluation plan (Task 1.2/1.4). 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 initial status of the software architecture for PowerStack path (Section 5.1) and workflow engine path (Section 5.2), which will be used as blueprints for the other work packages, e.g., the software prototyping in WP3 (Section 6). Further, this document also covers the current status for the evaluation plan (Section 7) and then 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.

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)
- **Section5.1 (PowerStack Path):** Eishi Arima (TUM)
- **Section5.2.1 (Melissa Path):** Bruno Raffin (UGA)
- **Section5.2.2 (Ryax Path):** Yiannis Georgiou (RYAX)
- **Section6 (Initial Ideas of PowerStack Integration):** Daniele Cesarini (CINECA), Eishi Arima (TUM)
- **Section7 (Evaluation Plan):** Georgios Goumas (ICCS), Nikela Papadopoulou (ICCS)
- **Section8 (Conclusions and Future Directions):** Eishi Arima (TUM), Georgios Goumas (ICCS)
- **Discussions/Fixes:** Abdelhafid Mazouz (ATOS), Millian Poquet (UGA), Georgios Goumas (ICCS), Eishi Arima (TUM)
- **Review Committee:** Andrea Bartolini (UNIBO), Pierre-François Dutot (UGA), Martin Schulz (TUM)

The following individuals joined our regular meetings for WP1 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), Julien Forot (ATOS), Yiannis Georgiou (RYAX), Georgios Goumas (ICCS, Lead), 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), Alessio Netti (BADW-LRZ), Michael Ott (BADW-LRZ), Nikela Papadopoulou (ICCS), Ioannis Plakas (UBITECH), Millian Poquet (UGA), Bruno Raffin (UGA), Olivier Richard (UGA), Martin Schulz (TUM), Mathieu Stoffel (ATOS), Carsten Trinitis (TUM), Xenofon Trompoukis (NTUA), Marianna Tzortzi (ICCS)

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 Nov. 2021 [2], has a few hundred-petascale computational capacity and a power consumption that can reach almost 30MW. Even with the highest technological advancements, an exascale machine is expected to well exceed the 20MW threshold that was initially set as the 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 (Section 5) instantiated with the use of the various modules brought in REGALE and evolved throughout the project by the partners (Section 6). To validate the efficacy of the REGALE approach and the alignment with the already set objectives and KPI's, we present our evaluation strategy in Section 7. Finally, Section 8 concludes this initial work and introduces several future research directions.

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 opower 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. Strawman Architecture and Software Tools

In this section, we first introduce the REGALE strawman 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.

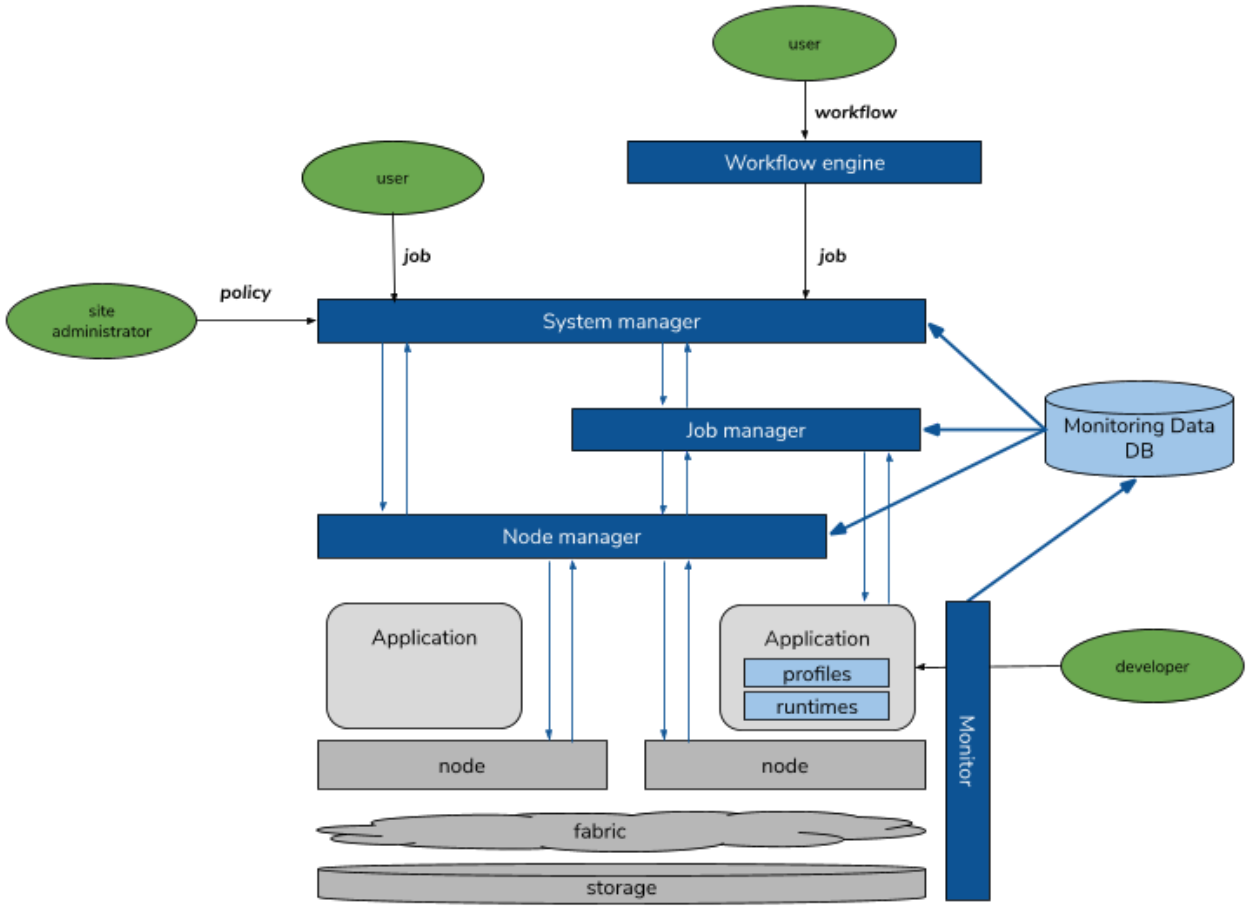

Figure 1: REGALE Strawman Architecture

Figure 1 illustrates the general strawman 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:

A. **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.

B. **Job manager:** The job manager performs 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.

C. **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.

D. **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.

E. **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.

To realize the software architecture, we integrate the following tools. TABLE 1 represents the general classifications of these tools into the system modules.

- **SLURM** is an open-source resource manager, with scheduling plugins, used by many supercomputers worldwide. While SLURM's main functionality is to allocate resources, start, execute and monitor jobs, it is extensible with plugins for scheduling, monitoring and accounting, and sophisticated resource allocation and job prioritization optimization methods. SLURM is designed to be highly-scalable, fault-tolerant, and portable.

- **DCDB** is a scalable monitoring framework for the acquisition of both in-band[1] and out-of-band[2] sensor data in HPC systems. DCDB follows a modular software architecture, allowing for the implementation of different plugins supporting a variety of data sources and protocols, and is currently deployed on BADW-LRZ's facilities. DCDB has been further extended with Wintermute, a data analytics framework, supporting online and on-demand data analysis.
- **BEO (Bull Energy optimizer)** is a tool for monitoring the power, energy, temperature and performance of the whole cluster infrastructure. Based on out-of-band monitoring through standard protocols (IPMI, SNMP, Redfish) and in-band monitoring (BDPO) and on a consolidated and distributed database, BEO can provide energy related insights without instrumentation. Connected with a system manager, BEO can provide detailed accounting metrics for a given job. BEO can be easily extended to support any additional hardware. BEO contains secured (OpenID connect, LDAP compatible) interfaces: CLI, Rest API (OpenAPI specification) and customizable web user interface intended for system administrators and end users.
- **BDPO (Bull Dynamic Power Optimizer)** is a job-oriented energy optimization tool/runtime system. It is integrated with SLURM and exposes BEO APIs. BDPO collects fine-grain performance-centric metrics, e.g., IPC and memory activity. It also performs precise in-band monitoring for efficiency metrics, like CPU energy consumption, in order to extend job data analysis provided by BEO. It offers an offline analysis mode, where applications are broken up into multiple representative patterns, highlighting the various recurring computations. These patterns are classified in terms of their energy footprint on the target hardware. BDPO can also dynamically adapt the CPU frequency based on these metrics in order to optimize the ratio performance/energy to solution.
- **OAR** (https://oar.imag.fr) is a versatile resource and task manager for HPC platforms, and other computing infrastructures. It is an European open source tool. OAR modular architecture makes it very flexible for integrating other tools or adding new scheduling policies.
- **Melissa** (https://melissa-sa.github.io/) is a file avoiding, adaptive, fault tolerant and elastic framework, enabling very efficient executions of ensemble runs on large scale supercomputers. The amount of storage needed for ensemble runs can quickly become overwhelming, with the associated long read time that makes statistic 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. Melissa bypasses this limitation by avoiding intermediate file storage. Melissa processes the data in transit enabling very large scale sensitivity analysis. Outputs are never stored on disc. This enable compute oblivious statistics maps on every mesh element for every timestep

[1] **In-band [8]:** Data sampled and consumed within a specific component in an HPC system, usually a compute node. Techniques using such data sources often operate at a fine temporal scale (i.e., greater than 1Hz) and require low analysis overhead and latency in gathering data.

[2] **Out-of-band [8]:** Data potentially coming from any available source in the system, including historical or asynchronous facility data. For techniques using this type of data, operation often has to be at coarse scale (e.g., in the order of minutes) and must be explicitly synchronized (e.g., through time-stamps), but latency and overhead are less of a concern. In this document, we classify the sensing/actuating via the board base controller into out-of-band.

on a full scale study. Experiments have demonstrated Melissa scalability up to 29,000 cores and processing of 273 TB on-line data.

- **RYAX** is a workflow management system for data analytics. It enables workflow composition and orchestration on hybrid distributed infrastructures such as HPC systems or the Cloud. It is built on top of Kubernetes inheriting its interoperability, flexibility, fault-tolerance and powerful declarative configuration. RYAX proposes a new DSL language that facilitates the definition of workflows. These architectural choices enable efficient stream processing, even though batch processing is possible, and support deploying on hybrid distributed environments. Task placement is delegated on the Kubernetes scheduler whose modularity allows us to develop different types of scheduling algorithms. RYAX facilitates the convergence of HPC/AI since it enables the execution of hybrid workflows through integration/communication with the dedicated HPC or Big Data resource managers. Kubernetes currently supports different types of containers platforms (Singularity, Docker) and this is maintained by RYAX.
- **BeBiDa** (https://gitlab.inria.fr/mmercier/bebida) is a resource management tool that enables the collocation of HPC and Big Data workloads leveraging the idle resources of an HPC system. The technique is seamless for end-users, it demands no change on the underlying resource management HPC system and is based on the simple job prolog/epilog mechanism, which is typical for HPC resource managers. The technique leverages Big Data frameworks resilience and elasticity by using a dynamic resource pool, and minimizes interference with HPC applications, since the Big Data applications are executed as low-priority best-effort jobs that get removed when an(other) HPC job needs the resources. BeBiDa ensures that no Big Data processes are left on the compute nodes after execution completion.
- **EAR**(https://www.bsc.es/research-and-development/software-and-apps/software-list/ear-energy-management-framework-hpc) is an open-source management framework optimizing the energy and efficiency of a cluster of interconnected nodes. To improve the energy of the cluster, EAR provides energy control, accounting, monitoring and optimization of both the running applications and the cluster. EAR is robust and reliable and has been in production on Supremacy-NG (LRZ) since 2019. At EAR's core there are two components: the EAR Daemon (EARD), and the EAR runtime (EARL). EARL is a dynamic, transparent and lightweight runtime library that optimizes and controls the energy consumed by MPI jobs without any application modification or user input. EARL dynamically identifies repetitive regions in parallel applications. The application information collected by EARL reports basic performance and power metrics. The application signature together with the system signature are inputs to the default power and time model used by EARL. EAR includes a plugin mechanism for power policies, which can be used by EARL to offer new policies. A power model allows the evaluation of new models and/or approaches for power/time projections such as neural networks. Energy accounting and power monitoring is provided by EARplug and EARD. EARD is a Linux service running with privileges in computing nodes. This service continuously monitors power and other relevant node metrics, such as temperature and average frequency, and reports them to the DB through EARDBD (an internal EAR component not extended in the REGALE project). EARD uses an energy plugin to provide energy

readings. By default, plugins for the Intel NodeManager and Lenovo SD650 used at LRZ are provided using the openipmi driver as well as a node energy estimation based on RAPL counters.

- **EXAMON** is an open-source monitoring framework deployed at CINECA and designed by UNIBO. It is composed of three main layers, the Data Collection, the Communication, the Storage and the application layer. The Data Collection layer samples two kinds of data i) the physical data measured with sensors and ii) workload information obtained from the job dispatcher. These software components are composed of two main objects, the Message Queue Telemetry Transport (MQTT) API and the Sensor API object. The Communication Layer is built around the MQTT protocol. The storage layer is based on a distributed and scalable time-series database (KairosDB) that is built on top of a NoSQL database (Apache Cassandra) as back-end. A specific MQTT subscriber (MQTT2Kairos) is implemented to provide a bridge between the MQTT protocol and the KairosDB data insertion mechanism. The Application Layer takes care of the data gathered by the monitoring framework, which can serve multiple purposes. For example, ML techniques can be applied to extract power/thermal predictive models or devise online fault detection.
- **COUNTDOWN** is an open-source runtime library that is able to identify and automatically reduce the power consumption of the computing elements during communication and synchronization of MPI-based applications. COUNTDOWN saves energy without imposing a significant performance penalty by lowering CPUs power consumption only during waiting times for which performance state transition overheads are negligible. This is done transparently to the user. Since COUNTDOWN targets performance-neutral energy savings, its goal is to avoid performance penalties for a large set of MPI-based applications. Thus, COUNTDOWN focuses on saving energy only when this has no effects on performance.
- **PULPcontroller** is an open-source HW/SW prototype for on-chip power management of HPC processors. It is developed as part of the European-Processor-Initiative (EPI) by UNIBO. It is composed of an open-source RISC-V HW design and an open-source firmware which implements the power capping, thermal management services as well as the interface with the operating system and with the Board Management Controller Firmware. The first realization of the PULPcontroller will be in the RheaR1 EPI chip. It can be emulated on FPGA boards.. Thanks to its open design and flexible architecture, it can also serve as a firmware level node manager.

TABLE 1: Tool Classifications

	Monitor + Database	Node Manager	Job Manager	System Manager	Workflow Engine
SLURM				X	
OAR				X	
DCDB	X				
BEO	X	X			
BDPO			X		
EAR	X	X	X		
Melissa					X
RYAX					X
Examon	X				
COUNTDOWN			X		
PULPcontroller		X			
BeBiDa				X	

We realize different implementations using the above tools based on the REGALE strawman 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 (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 Task 1.1, we gathered the requirements posed by all key actors in the REGALE architecture, with a particular focus on the PowerStack path as an initial step. Note that, in the future deliverables, we will comprehensively cover all the paths including Melissa and RYAX paths (see also Section 8). 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 (see also Section 6 for the initial tool assessment).

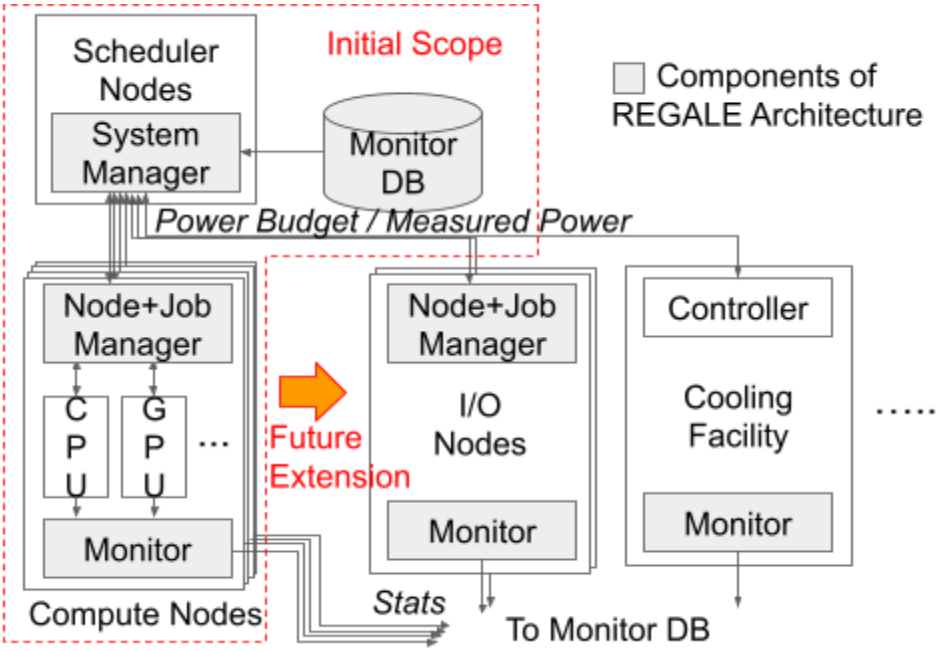

Figure 2: Holistic Power Management and Our Current Scope

4.1 Our Initial 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 manager daemon launched on the scheduler (or admin) nodes. More specifically, the system manager 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. As an initial step, 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 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.

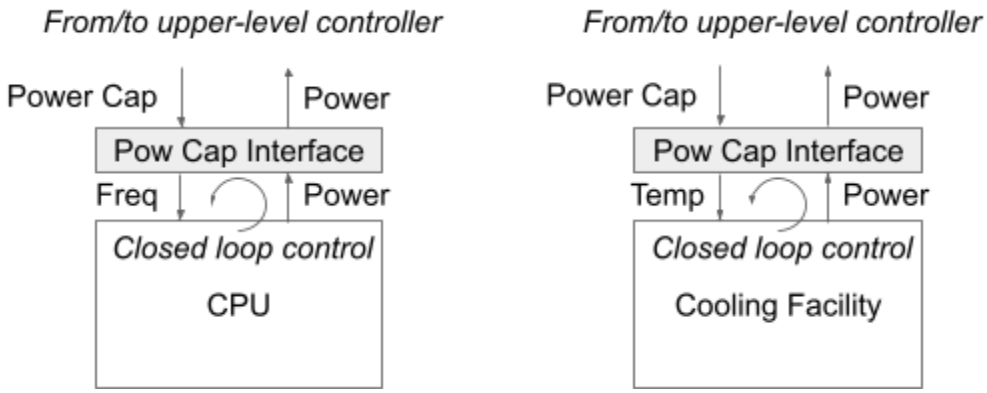

Figure 3: Power Capping Interface Requirements

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 power management knobs in accordance with the monitored power, in order to enforce any given power cap; and (3) the interface must also provide the actual power consumption. 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.

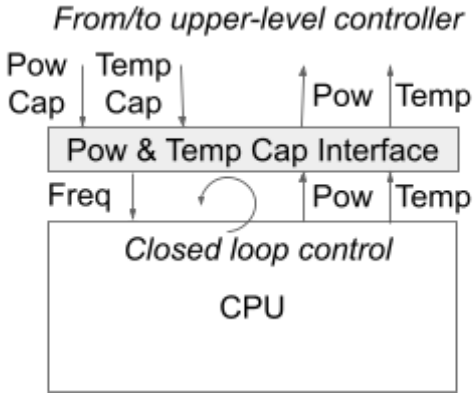

Figure 4: Interface Extension for Thermal Capping

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) are the components that manage the power control optimization. They can be at system level, at application level, or a combination of both.

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**
4. **Job Manager**
5. **Node Manager**
6. **Monitor**

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, 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 sophistications

	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:** 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:** 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

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:** 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

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.

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;	**Objectives:** Max system throughput (Level2) **Constraints:** Power cap (Level1) – or extensible to include more (Level2/3) **Knobs:** CPU power cap (Level1) **Temporal Granularity:** Dynamically adjusted at runtime (Level3)	**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:** All the functionalities supported in Basic; Periodical power budget redistribution function based on the reported unused power and

Assuming over provisioned situation; Adding more constraints is an option (e,g., thermal cap, application speed-down limit)	**Spatial Granularity:** Inter node optimization (Level2) **Optimizers:** System-level optimization (Level1)	power budget request by Node Manager; Power budget distribution policy to maximize throughput **Node Manager:** All the functionalities supported in Basic; Periodically measure power and application stats at runtime (PMU, sysfs); policy to detect whether the node needs less/more power budget; report the above to System Manager **Job Manager:** None **Monitor:** Providing monitoring data to Node Manager (option)
[**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:** 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:** 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 (option)
[**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	**Objectives:** Max system throughput (Level2) **Constraints:** Power cap (Level1) – or extensible to include more (Level2/3) **Knobs:** CPU power mnmt knob (Level1)	**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

is an option (e,g., thermal cap)	**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)	**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 **Job Manager:** App profiling to optimize power mgmt knobs (power cap, clock freq, etc.) as well as a power-aware code tuning functionality **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 total system throughput under power cap 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 (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:** Dynamically adjusted at runtime (Level3) **Spatial Granularity:** Inter- and intra-node optimization (Level3) **Optimizers:** System-level optimization (Level1)	**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)	**Objectives:** Max system throughput (Level2) **Constraints:** Power cap (Level1) – or extensible to include more (Level2/3)	**Site Admin:** Same as NodPowShft **Users/Developers:** Same as NodPowShft

Note: Should be agnostic in objectives and constraints; Needs job characteristics estimation using historical data collected by Monitor; Here we assume this use case is an extension of NodPowShft.	**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) **Optimizers:** System-level optimization (Level1)	**System Manager:** Power-aware scheduling policy using the historical job statistics **Node Manager:** Same as NodPowShft **Job Manager:** Same as NodPowShft **Monitor:** Providing collected job statistics under power optimizations

4.3.4 Discussions

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 as shown in TABLE 2. We then describe the necessary functionalities/interfaces to realize these use cases in Section 5.1 while reflecting the current state of our software tools. Section 6 summarizes our initial step for integrating these software tools to enable use cases described here. Further, we also introduce more sophisticated use cases in the future deliverables based on the studies to be conducted in WP2.

We will continuously assess and update the use cases in terms of necessity, coverage, comprehensiveness, and so forth throughout the project. In this document, we picked up these use cases, starting from the most naive one and sophisticating it step-by-step toward different directions, while considering the current tool support. For this, we take multiple different aspects (e.g., objective, constraint, etc.) into account to define a use case, while setting up multiple levels for each of them. If this setup is appropriate, we can assure the necessity, coverage, and comprehensiveness of this approach. We will continuously check the appropriateness with tool developers, HPC users, or site admins, not limited within the project but also includes outside communities such as the PowerStack initiatives.

As for the tool support for each use case, we will start implementing from the most naive one (i.e., the Basic use case) and will gradually increase the coverage as well. We will at least consider realizing all the use cases listed here (and more options will be explored in WP2), and if one is turned out to be non-relevant to the project, we will update or remove it accordingly.

5. Architecture Descriptions

In this section, we explain the details of the initial architecture for different paths: PowerStack path and workflow engine path. The former is based on the PowerStack use cases specified in the last section, and we accordingly extend the strawman architecture (see also Figure 1). The latter consists of two different workflow engines our project has, Melissa and RYAX, and we describe the software architectures of these tools.

5.1. PowerStack Path

Following our strawman architecture and the PowerStack use cases described in the previous sections, we define the details of the software architecture for the PowerStack path. Before delving into the details, we summarize the roles of the use cases and the software architecture in the overall PowerStack development and the relationship with other work packages.

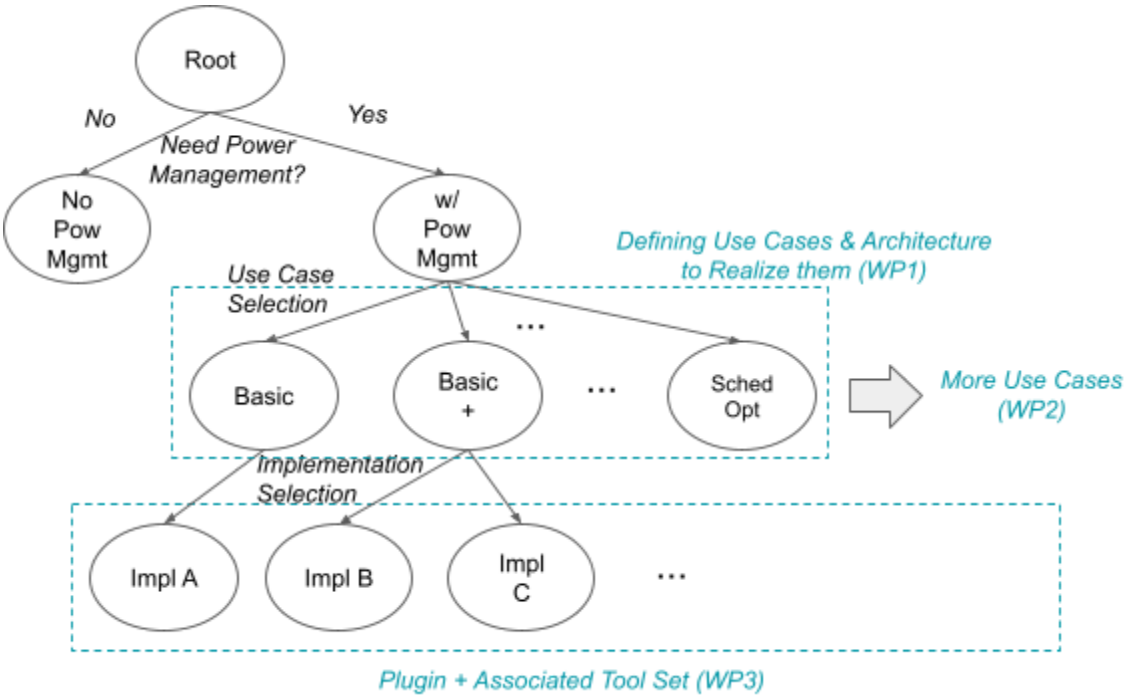

Figure 5: Overall Decision Tree for Site Administrator

Figure 5 represents the decision tree a site administrator would follow when utilizing REGALE for resource management. There should be different options for the power management, and the site administrator would choose one of them, which fits the site policy the best. One or more implementations can exist for one use case (e.g., implemented with different software tools or different algorithms to realize it), and when an option is designated by the site administrator, the set of associate plugins or tools should be selected and work coordinatively. This actual implementation/integration part is the role of WP3, and the initial idea is described in Section 6. In contrast to this, the software architecture is an open and general form of describing the

required functionalities each software component should have to realize a certain use case, which should not be tool specific, and for this, we follow the requirement specifications written in Section 4. However, even so, we also collected the current states of software tools we have in the REGALE project and checked if the direction of software architecture fits them well. In WP2, we are investigating a variety of sophistication techniques for improving resource management, some of which would contribute to enhancing the use cases and to extending the functionalities in the final PowerStack implementation.

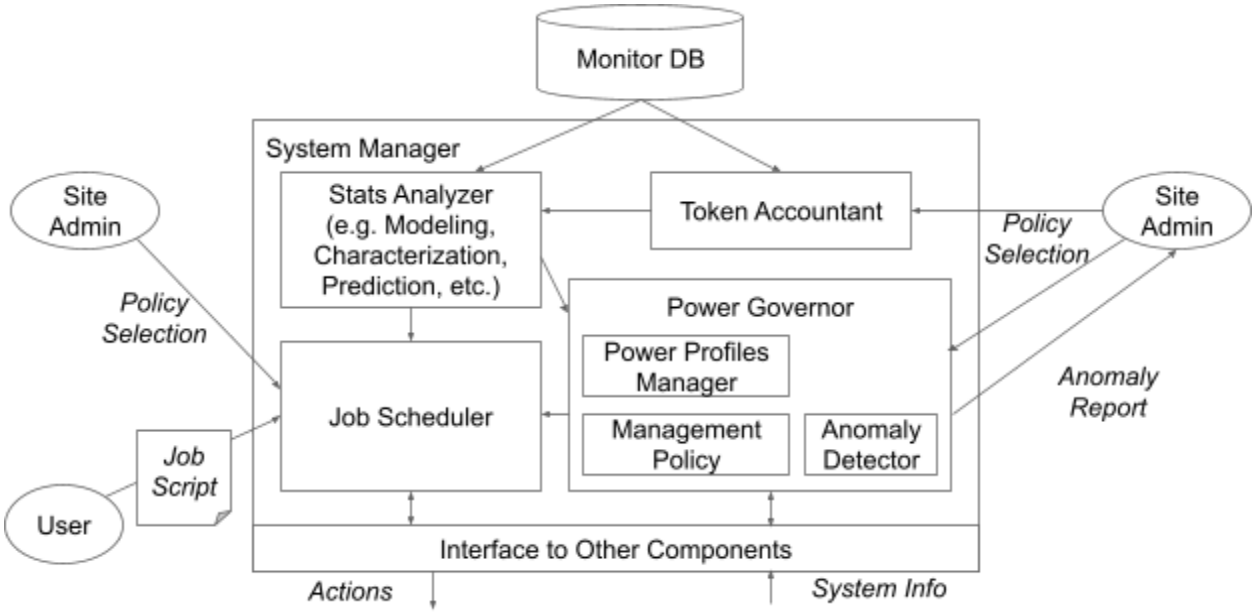

Figure 6: Software Architecture of System Manager

We then introduce the software architecture of System Manager based on the requirements specified in the last section – Figure 6 illustrates the architecture. This component consists of multiple sub-modules, and the roles of them are listed as follows:

- **The Power Governor** makes decisions on the power budget distribution as the central controller. This sub-component provides several **power management policies** associated with the use cases, and one of them is selected by the site administrator. In practice, these options are implemented as plugins in a system management tool such as Slurm. This sub-component should also own the **power profile manager** that collects a variety of hardware information used in these power management policies. For instance, the Basic use case needs to know the upper limit of system power consumption when the CPU power capping is applied across all compute nodes, and if the system power cap designated by the site administrator is out of the applicable range, that should be reported. Another example is that the profiler should also know which components are the targets of the power management. Further, the power governor should have the **anomaly detector** that gathers anomaly behaviors reported by the other components (or detects by itself from reported power consumption) and notify them to the site administrator (or any other components if needed).

- **The Stats Analyzer** is mainly used for the sophisticated resource management techniques to be explored in WP2. This sub-component analyzes the historical statistics collected by the Monitor to gain hints/insights into the scheduling decisions, such as the characteristics of applications, the job submission pattern, which is a function of data like system information, predicting the future system state based on them, and so forth. The results of analyses here should be accessible to the Job Scheduler so that it can use these functionalities. The exact functions required here depend highly on the scheduling objectives, methodologies, algorithms, etc., which are going to be explored in WP2.
- **The Job Scheduler** can be one of the main actors for our power management, particularly in the SchedOpt use case. This sub-component attempts to schedule jobs so that the objective function is maximized/minimized under given power or thermal constraints. In SchedOpt use case, the Job Scheduler needs to know the following aspects: (1) the characteristics of all queuing jobs, which are supplied by the associated job scripts or the Stats Analyzer depending on the information; (2) the power management policy currently applied in the Power Governor; and (3) the status of currently running jobs including the actual power consumption.
- **The Token Accountant** is responsible for calculating the token consumption, i.e., how much the site charges for each executed job, which is a key factor and needs to be revisited for several use cases. 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. Thanks to the monitoring tools, we can realize this option in the REGALE project.
- **The Interface to Other Components** is used for sending out the decisions made by the system manager such as launching a job and redistributing the power budgets and also for collecting system information needed for the decision making including actual power consumption, anomaly reports, power requests, and so forth.
- **Human Actors: The Site Administrator** selects the policies in the Job Scheduler and the Token Accountant, and the Power Governor and takes an action when an anomaly is reported accordingly. **Users** submit their jobs to the job scheduler. In addition, for several use cases related to job-oriented power/energy optimizations, they care about the power/energy consumption in their jobs. For this, they may link relevant Job Manager libraries to their codes and set up several parameters or variables accordingly to take advantage of the PowerStack software tools. In addition, applying power- or energy-aware code optimizations to their application by hand or in an automatic manner is an option for them for one step further optimizations. For this, the site administrator may allow the users to access the power management knobs depending on their policy.

Next, we describe our system-level node power management using the Node Manager and the Monitor component in Figure 7. Here are the details:

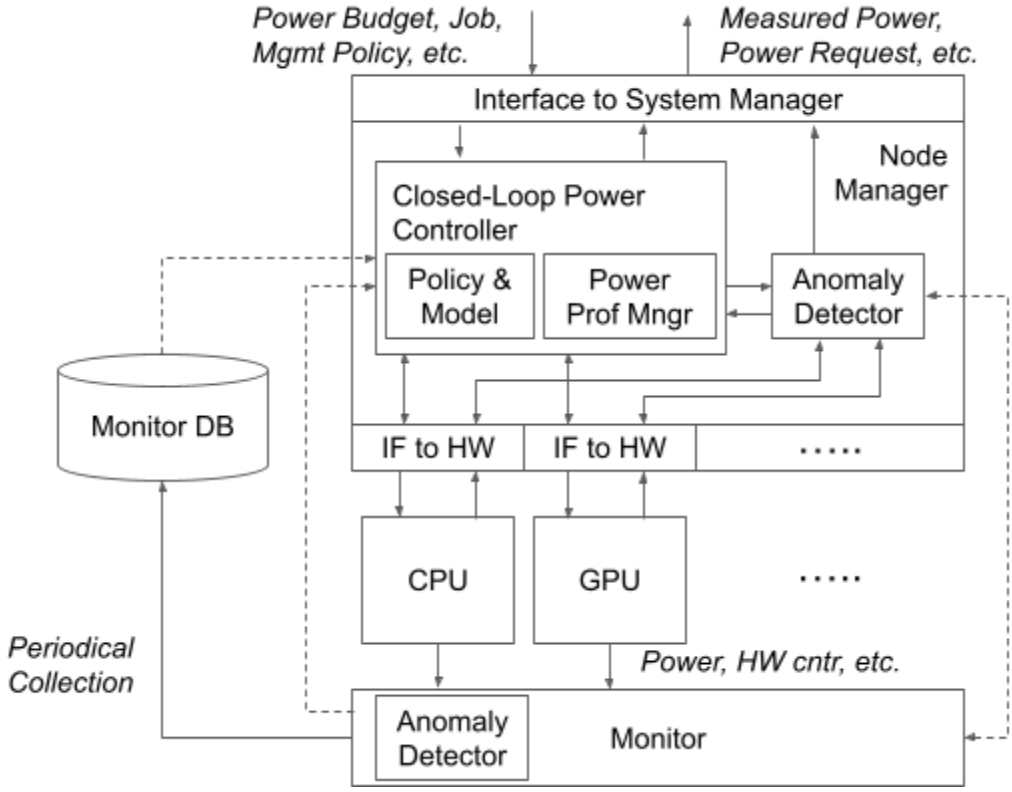

Figure 7: System-level Node Power Management

- The Node Manager should be able to access the in-band or out-of-band hardware sensors and actuators via the relevant interfaces that meet the requirements specified in the last section. As an option, it will enhance the hardware sensor accessibility by using the API or the monitor database, the monitoring tools offer.
- The Node Manager should also have an interface to interact with the System Manager. As described in the last section, we assume the Node Manager receives power management information sent by the System Manager, such as the node power budget, the power management policy associated with the selected use case, etc.. It also periodically sends the node status/requests including the measured node power, the request for less/more power budget, the detected anomaly, and so forth.
- The Node Manager has a **Closed-Loop Power Controller** that controls power management knobs in accordance with the measured data at runtime. As described in the last section, the power measurement is a requirement, and collecting other runtime information such as hardware performance counters is an option. Another option is using historical data previously obtained by the Monitor. This Closed-Loop Power Controller should have a **Power Profiles Manager** that calibrates/trains the hardware related parameters in the power/performance models used in the control loop. The exact models, algorithm, and policy used in the power management should be provided as a plugin.

- The Node Manager should also have an **Anomaly Detector** that detects an anomaly state by comparing the measured power and the power limit set by the controller. If a significant power cap violation happens, it should report that to the System Manager and also handle it accordingly while interacting with the controller. This component is independent of the controller as this functionality should work regardless of the power management policy currently working. The exact error handling policy is up to the use case or the software implementation. Further, this function can sit inside of the monitoring tool so that it can record the anomaly information on the database.

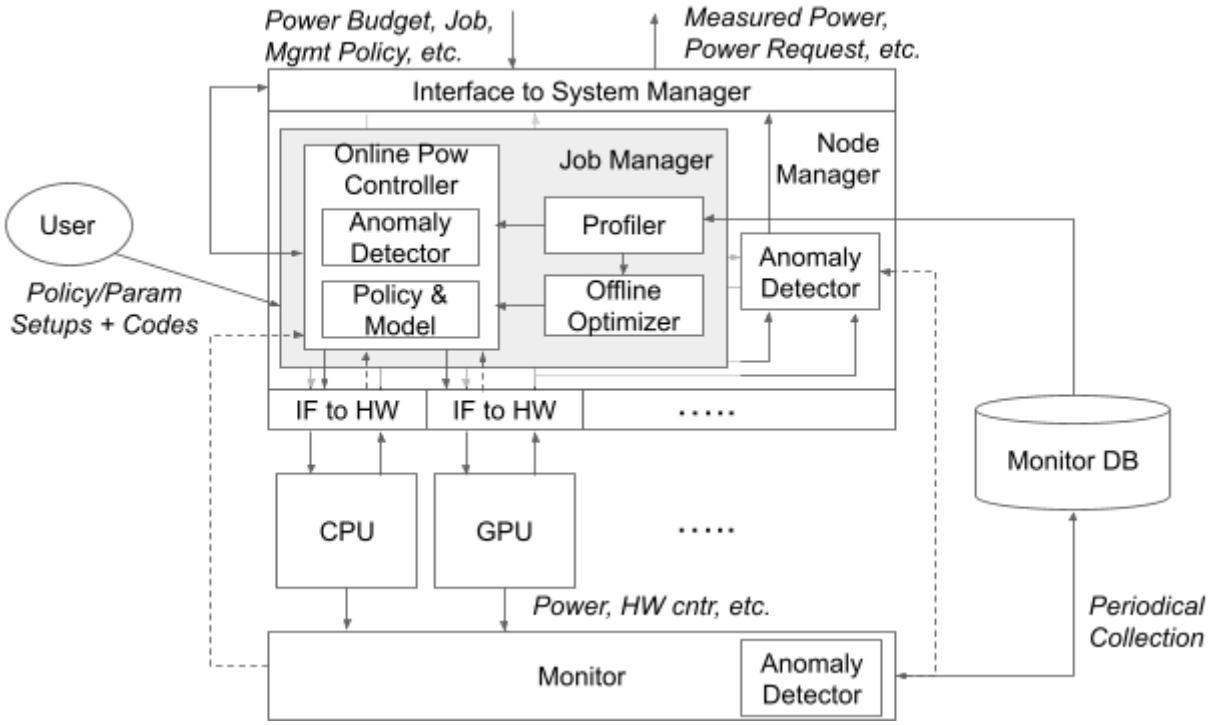

Figure 8: Application-level Node Power Management

Finally, Figure 8 illustrates our application-level power management at a node. The Job Manager plays a key role here and takes over some tasks from the Node Manager to optimize power management knobs in a more application-aware fashion. For this, the Node Manager should be able to provide an interface so that the Job Manager can utilize some of the functionalities such as interacting with the System Manager (e.g. by using SLURM APIs), and also should be able to arbitrate/prioritize the power management knob accesses between them. The Anomaly Detector in the Node Manager should operate even with the Job Manager and can enforce strict power management knob setups if an anomaly is detected. Note the actual division of labour depends on the given use case or the implementation. As an example, if the thermal constraint is applied, yet, the Job Manager doesn't support this feature, the Node Manager should handle it, i.e., it will take over the power management knob control when the temperature violation happens. The following list describes the sub-components inside of the Job Manager:

- **Users** (or developers) can link the Job Manager libraries to their codes when compiling them, in order to utilize the power management functionalities. They can also change the policy or parameter setups in the Job Manager, e.g., via configuration files. They may also be able to control the power management knobs by themselves through the APIs of the Job/Node Managers or lower-level interfaces, which is however usually not permitted by the site administrator due to the security concerns.
- **The Profiler** accesses the Monitor Database and analyzes the profiles of previous runs of the application or any others in order to help with the power management decisions. One role of the Profiler is to characterize the target applications by using several statistics, including compute intensity, cache miss rate, branch prediction miss rate, or any others. For instance, BEPO attempts to detect the phases of the target application by using some of these metrics. These metrics can also be used as inputs to models to estimate performance, power, or energy for several tools. Another role of the component is determining the coefficients in those models, which are usually functions of hardware characteristics. To train the coefficients, it can also utilize the profiles of previous test runs of this application or a pre-selected benchmark suite. The actual training methodology depends highly on what models we use here – for instance, the naive curve fitting is used for the widely-used linear regression modeling. Further, as an option, the profiler may associate performance, power, or energy with different combinations of code optimizations (e.g., compiler flags) and power management knob setups for the target application, which is useful for the autotuning purpose.
- **The Offline Optimizer** is responsible for any static and proactive decisions made by the Job Manager. If the tool determines the power management knob setups before running the job, this offline optimizer handles the setups. Here, the setups should be determined for all the possible node power budgets. Another example of static decision is selecting the models that fit the best for the target application from given variants. Further, another option is as described above, choosing an optimal set of codes as well as the power management knob decisions.
- **The Online Power Controller** handles the power management knobs at runtime based on the static decisions made by the Offline Optimizer. It should also be able to access the runtime information, such as current node power budget, to select optimal power management knob setups. As an option, the Job Manager can work in a more dynamic manner, such as a closed-loop power control using runtime application statistics. In either case, optimizer should be defined to be able to tolerate input dependencies, as the characteristics or behaviors of some applications highly depend on their arguments or input files. The power controller may have its own anomaly detection function.

5.2 Workflow Engines

5.2.1 Melissa Path

Melissa is a framework dedicated to the on-line data processing of large scale ensemble runs. Ensemble runs consist in running several instances of usually the same simulation code, with different parameters. This kind of approach is also referred to as parameter sweep or Monte Carlo methods. They are getting more common as the compute power available is increasing. Methods relying on ensemble runs include sensibility analysis, training of deep surrogate models, data assimilation, reinforcement learning to name a few. The amount of data generated by an ensemble run, that can consist of thousands of simulation runs, each one being already a complex advance parallel simulation code, is quickly overwhelming, saturating the system storage, slowing down the simulation executions, but also other users' runs impacted by interference on the I/O system. On-line data processing solutions are thus critical.

Melissa (https://gitlab.inria.fr/melissa) is an open source (BSD) framework dedicated to large scale ensemble runs and on-line/in transit data processing. Melissa has been developed with built-in features that are essential when targeting Exascale: fault tolerance and elasticity. As an example of Melissa capabilities, largest runs for sensibility analysis study so far involved up to 30k core, executed 80 000 parallel simulations, and generated 288 TB of intermediate data that did not need to be stored on the file system.

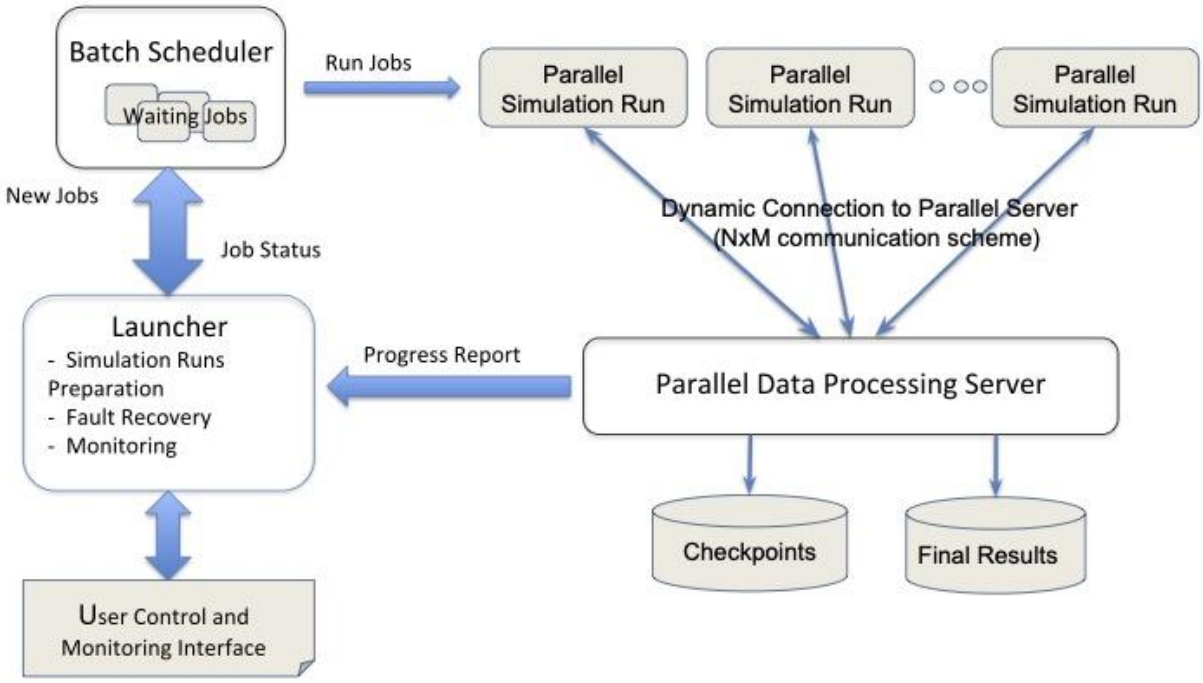

Figure 9: Melissa Architecture Overview

Melissa architecture relies on 3 interacting components:

- **Melissa runners (clients)**: the parallel numerical simulation code turned into a client. Each client sends its output to the server as soon as available. Clients are independent jobs.
- **Melissa server**: a parallel executable in charge of iterative (on-line) data processing. Upon reception of new data from any one of the connected clients, the server processes it to update its on-line computations and discard it, sot the memory requirements on the server side stay under control.
- **Melissa Launcher**: the front-end Python script in charge of orchestrating the execution of the ensemble run. This is the user entry point to configure the study of experiment. The launcher is also tightly interacting with the machine batch scheduler to control executions, including faults and elasticity.

Melissa today supports directly 2 types of applications, plus a third one in development:

- **Sensibility analysis** where the parallel server compute statistics relying on parallel incremental algorithms (for computing the average, variance and co-variance, skewness, kurtosis, minimum, maximum, threshold exceedance, quantiles and Sobol' indices)
- **Data assimilation** where the server implements a parallel EnkF filter or a particle filter, with capabilities for load balancing member propagations to clients.
- **Deep surrogate training** where the parallel server trains a deep neural network from the data produced by the client. Training is performed in parallel on several GPUs using model parallelism through the Horovod library.

Melissa tightly interacts with the batch scheduler as each client as well as the server run independent jobs. Melissa's work with the batch scheduler consists in submitting jobs, monitoring their status, and killing and restarting failed ones under the control of a fault tolerance protocol. Melissa is also both moldable and malleable:

- Malleability (elasticity): clients can be killed or added at any time during the execution, if under external requests the melissa application is requested to free resources or given the possibility to run more clients. We often refer to this capability as elasticity.
- Moldability (heterogeneous clients): different client instances can run on different resources as each one at connection time setup its own NxM communication pattern with the server. Melissa internal load balancing support also enables these clients to progress at different speeds. So some clients may for instance use GPUs and other be running on CPUs only.

In the context of the Regale project Melissa is expected to be used for Pilots 1, 2 and 5 for sensibility analysis and deep surrogate training. These pilots, in addition to Melissa already existing use cases, will be part of the benchmarking suite to experiment various resource allocations strategies.

Melissa will evolve to leverage the tools and features developed in REGALE for resource allocation and monitoring:

- The Melissa launcher will be augmented to provide the necessary information to the batch scheduler so this last one can:
  - Identify that the Melissa jobs belong to the same run.
  - Mold each job according to resources available, potentially recognizing that client jobs all have similar execution profiles, opening the possibility to infer from past clients execution the behavior of the upcoming clients to execute.
  - Adapt on-line the resource used, potentially killing some clients jobs if necessary (malleability).
- The Melissa launcher will be augmented to aggregate and report to the user the power consumption (global, per job, etc.), relying on Regale power monitoring tools.

5.2.2 RYAX Path

The Ryax platform developed by Ryax Technologies is a proprietary workflow management system for data analytics. It provides the means to create, deploy, update, execute and monitor the execution of data processing workflows on hybrid on-premises, Cloud, HPC computing infrastructures.

The Workflow Management system is responsible for the automation of orchestration and execution of task collections upon computational resources. A common pattern in scientific and cloud computing involves the execution of different computational and data manipulation tasks which are usually coupled i.e., output of one task used as input on another. Hence, coordination is required to satisfy data dependencies. The task execution is handled by the system and can be distributed among the underlying available computational resources. Consequently, this introduces further complexity on the system side, related to processes such as load balancing, data storage, data transfer, tasks monitoring and fault-tolerance. Furthermore, on the application side, workflows provide an end-to-end view of the processing rather than focusing on a specific part of the computation which allows users to control the full process by abstracting the complexity of how each task is executed. Automation of the aforementioned aspects of the orchestration process along with the complexity abstraction has led to the creation of workflow management systems.

The RYAX platform proposes a new DSL language based on YAML and particular abstractions that allow users to develop workflows of data processing taking into account the end-to-end dataflow view of the processes. As such, the design of a workflow which can be done either graphically through a Web User Interface or programmatically through a Command Line Interface incorporates the inputs and outputs of each task and how they are interconnected among them in DAG-style representations. Ryax, through its design and abstractions offers a

unified handling of batched and streaming data definitions to cover processing of both bounded and unbounded datasets, which makes it an attractive engine for hybrid Big Data/HPC applications.

In more detail the principal concepts of RYAX are:

- **A module,** which is an independent task representing a separate building block of a broader data processing application. It may have inputs, outputs and a specific code that manipulates the inputs with some logic to produce the outputs. There are **four types of modules** in RYAX:
  - **A source**, which allows data ingestion from the outside world. They are long running processes (micro-services) which can trigger new workflow executions.
  - **A processor**, which is a stateless process (FaaS) that basically *can perform processing*. They use their inputs to ingest data from upstream produce outputs added to the down stream.
  - **A stream Operator,** which is a specialized process to manipulate data streams hence addressing streaming analytics. For example, you may merge multiple data streams together or buffer data at any given point.
  - **A publisher**, which is a stateless process that pushes data to external services, like a database or an online service for example.
- T**he workflow,** which is a complete data processing application composed of modules linked together in the form of a DAG. The intermediate links are data streams. Each module uses some data from the input stream and adds its output data to the stream. This way, the data that a module outputs is accessible to every downstream module. In other words, any module has access to the data of upstream modules.

The internal architecture of RYAX is shown in Figure 10 and as we can see it is composed of different components managed by the Kubernetes orchestrator to guarantee interoperability, flexibility, auto-healing, easy upgrade/downgrade and fault-tolerance of the deployed components' microservices.

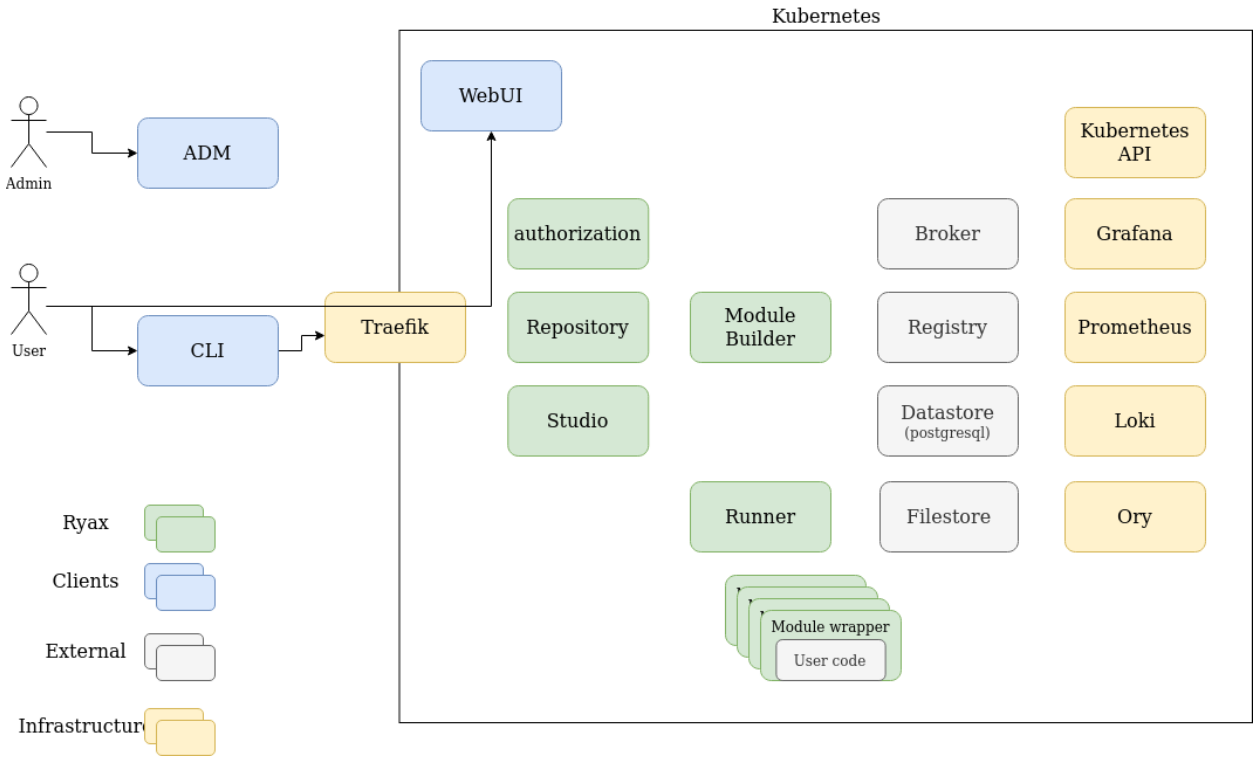

Figure 10: Ryax Architecture Overview

In particular we have:

- The Ryax client services:
  - **WebUI,** which is an NGINX server that serves our frontend written with Angular.
  - **ADM,** which is the administrator tool for Ryax used to deploy, update and backup/restore a Ryax instance upon a certain infrastructure.
  - **CLI,** which is an open-source Python based command line interface that enables users to build modules and create, submit and monitor workflows.
- The Ryax internal services:
  - **Authorization,** which manages user and project authorization along with roles control within Ryax.
  - **Repository,** allows the Ryax users to scan Git repositories and import Ryax modules. It also enables the triggering of the modules built through the Module Builder. Once the build is finished, the modules are sent to the Studio to be placed in the Module Store. It exposes an HTTP Rest API that is used by the WebUI and the CLI. It uses the Postgres datastore to persist its state using ORM.
  - **Studio,** enables Ryax users to create and deploy workflows. It exposes an HTTP Rest API that is used by the WebUI and the CLI. It uses the Postgres datastore to persist its state using ORM.
  - **Module builder,** which receives module build orders from the Repository service, does the build in a sequential and synchronous way (one at a time) making use of the NIX functional package management software for dependencies control.

    - **Runner,** which plays the role of interface between Ryax and the computing resources to launch and execute modules & run workflows. Thus, it enables the deployment, run and schedule of each module of the workflow.
      It also manages the execution metadata and data. This micro-service communicates with the modules through gRPC, http and the broker (RabbitMQ).
    - **Module wrapper,** which is used between our system and the user code to be able to run it. This is not an internal Ryax service, but a wrapper that is put around user code in order to communicate with Ryax. It creates a gRPC server with a simple interface that initiates the module and then run executions.
      This wrapper works for both processors and source modules with the same protocol: the source modules are streaming execution response, while the processors are only sending one response and close the connection.
- The Ryax external services:
    - **Datastore,** which enables the state storage of all statefull services and is based upon PostgresQL. Each service has a different access credential and a separate database.
    - **Filestore,** which is used to store execution I/O files and directories based upon a Minio file storage service which exposes an S3 compatible API.
    - **Broker,** which enables internal communication between all services using messages serialized in Protobuf and is based upon RabbitMQ message broker.
    - **Registry,** which stores the users' modules created by the Module builder in a Docker registry.

By default the architecture allows the execution of modules using the runner upon the underlying Kubernetes cluster on which RYAX is deployed. However, the Runner is currently being adapted in such a way that enables a finely integrated “on-demand offloading” to external Task executors, Orchestrators or Resource Managers, such as HPC schedulers like SLURM and OAR.

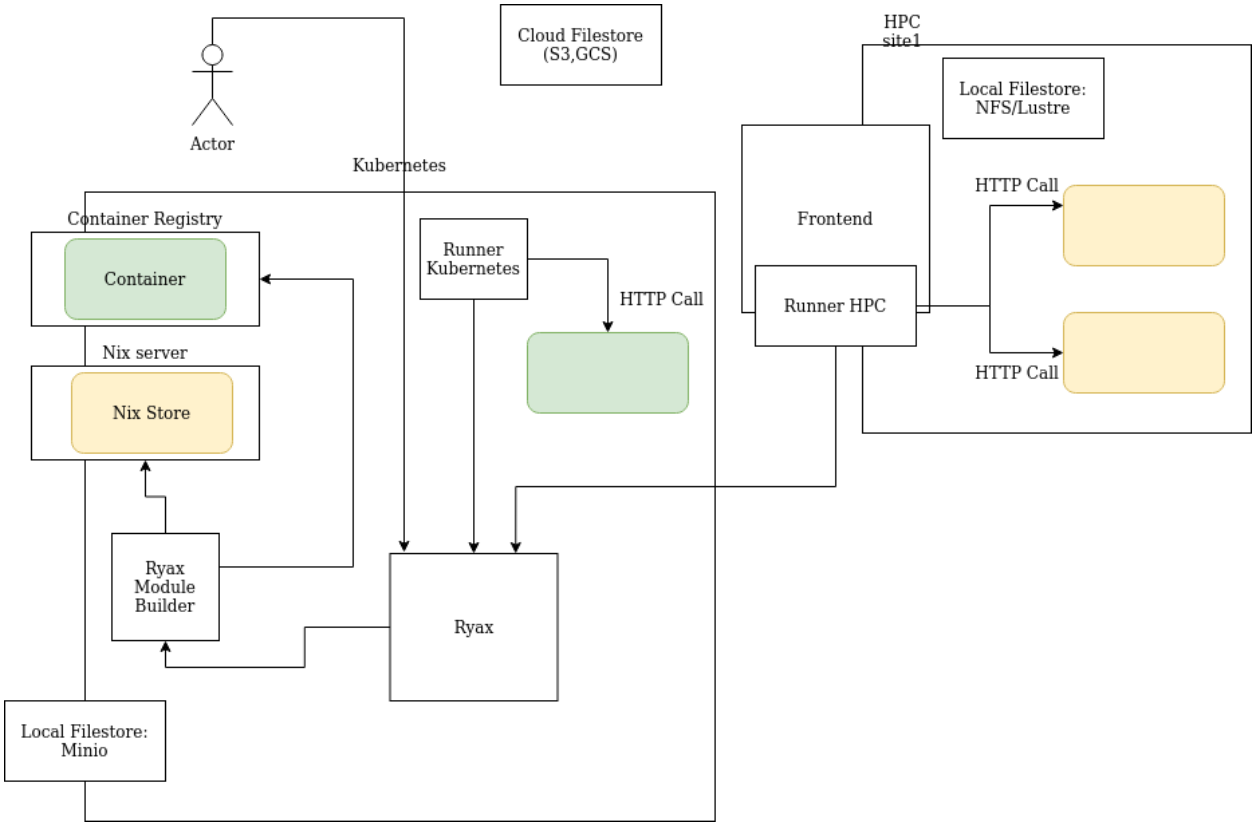

Figure 11: Ryax Architecture of On-demand Offloading to HPC infrastructures

Hence one of our contributions within the REGALE project is to adapt the Runner in such a way in order to enable the execution of specific modules upon particular resources of the HPC clusters. In particular users will have the ability to demand resources, select the HPC batch scheduler to use and enable the execution of a workflow so that modules can be executed on the connected HPC clusters.

In more detail our contributions in the context of Regale project will be:

- Support of SLURM (and OAR if time allows) batch scheduler(s) through an SSH or REST based execution mechanism
- Support of Singuarlity as HPC containerization runtime.
- Support of NFS as shared file system for data exchange among the modules being executed upon the HPC platform

The Ryax platform can specifically be used to deploy Big Data and ML applications on HPC clusters. In this context, besides the direct execution of Big Data applications using dedicated HPC resources, we will enable an integration with BeBiDa software to allow a more elastic execution of Big Data applications through the technique of Best-Effort jobs of batch schedulers. Figure 11 shows a high-level view of the integration between Ryax and BeBiDa.

In more detail our contributions in the context of Big Data applications will be:

- Support of the Spark Big Data framework
- Integration with BeBiDa software for elastic HPC resource management and improved Big Data execution guarantees.
- Multi-criteria scheduling techniques on top of Kubernetes to take into account HPC observability (resource usage, energy consumption, etc) to intelligently deploy the different modules of the submitted workflows.

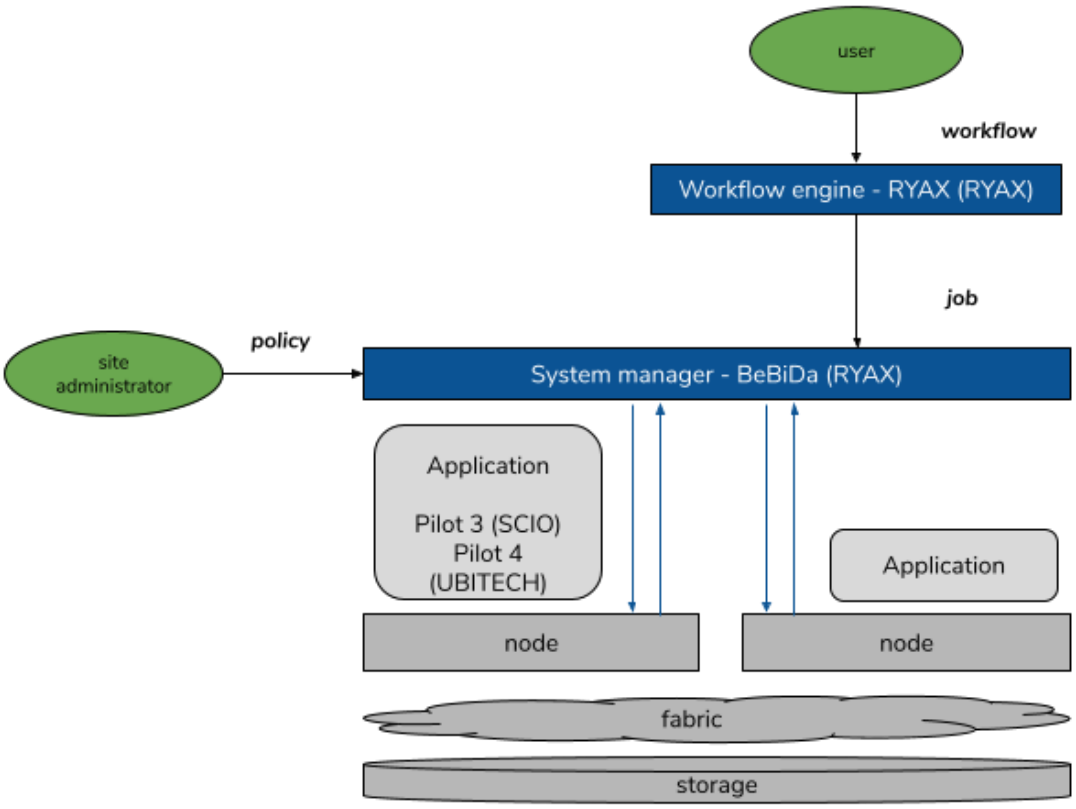

Figure 12: Ryax - BeBiDa integration

6. Initial Ideas of PowerStack Integration

In this section, we first assess our software tools to realize the PowerStack path. More specifically, 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 introduce our initial integration plan currently discussed in WP3 as well.

6.1 Tool Assessment for PowerStack

TABLE 8: 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				
	Job	Token	Power governor	Analyzer	API/Lib	In-band	Out-of-band	Anomaly	DB	API/Lib
SLURM	X	X	X		/	/		X	/	/
DCDB				X	/	/	/	X	/	/
Examon				X	/	/	/	X	/	/
BEO						/	X	X		/
EAR			X		/	/		X	/	/

TABLE 8 lists the candidate tools for System Manager and Monitor, and the current support for the key functionalities. OAR, a job scheduling tool, is not listed here as we consider SLURM as the core component in the PowerStack path due to the current support status, but OAR will be used for the workflow engine path or some sophistications in WP2. By default, SLURM has several key functionalities, including job scheduling, token management, and power management, but they may require updates depending on the use cases to support. For instance, we need to develop exact algorithms for the SchedOpt use case and to implement them as SLURM plugins. SLURM has a token management component as well, and extending it to support power-/energy-aware token management policy is needed for the AppEtS use case. As for the Power Governor, several basic power management functionalities/interfaces are already supported in SLURM, such as designating CPU power cap to each node/job, but we need to modify them to support more complicated use cases. Using EAR (EARGM) as the power governor is another option, and 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 may be updated for realizing several sophisticated use cases in the future (e.g. system modeling for cooling-aware power and job scheduling).

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 candidates 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 9: 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	Anomaly	Access to sys mngr	API/Lib	Profiler	Pow ctr + Anomaly	API/Lib	In-band	Out-of-band
SLURM	X	X	/	/				/	
PULP ctr	/	X						/	X
BEO	X	X	X	/				/	X
BDPO					/	X	/	/	
EAR		X	/	/	/	X	/	/	
Countdown						X	/	/	

Next, TABLE 9 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 Initial Integration Plan in WP3

One of the main goals of the project is the tool integration to realize a European software stack for power and workflow management in supercomputers of the new generation. The purpose of the WP3 is to accomplish this goal. This WP is started in November 2021 at Month 7 and it will conclude at the end of Month 36 with the conclusion of the project. All the partners involved in the WP3 focused the first month of work to explore an initial integration of the tools. In particular, most of the time spent in the WP3 during November and in the second plenary meeting was based on sharing integration ideas and possible points of contacts among all the tools involved in the project.

In TABLE 10, we presented most of the integration ideas and developments planned in the project. We highlight the improvement(s) on the tools that each partner will focus on during the project. In REGALE we will continue the work on the development and the extension of each tool to improve its maturity but, at the same time, each partner planned to integrate its tool with other tools proposed in the project. In particular, the TABLE 10 proposed two levels of integration, the first one is the planned integration, this work has been planned and described in the DoA document and it will be completed before the end of the project, it has been identified in the table with a *X*. The second is a potential integration discussed in the first month of work and represented in the table with a question mark. This will be discussed during the project to find a potential point of contact and the consequently integration. The goal of the project is to integrate as much as possible the tools to propose a larger and more efficient software stack. This is not always possible astools overlap in some functionality in particular on monitoring and power management. However we think that there are several points of contact to work on a broader integration to realize more efficient and mature tools. We show that this is possible in the TABLE 10, where we can see that there are several partners who propose a huge number of potential integrations that will be explored in the next project months in the WP3. However, this would go beyond what the project promised in the proposal to show that it is possible to make a more efficient European software stack for power and workflow management.

TABLE10: Matrix of tool integration that reports the integration discussed in the proposal
***I* = Tool improvement, *X* = Planned integration, *?* = Potential integration**

	SLURM	OAR	DCDB	BEO	BDPO	EAR	Melissa	RYAX	EXA MON	CNTD	PULP Ctr	Bebida
SLURM	-				*I*	*I*	*I*	*x*	*X*			
OAR		*I*			*?*	*X*	*I*	*?*	*?*	*?*		*X*
DCDB			*I*			*X*				*?*	*?*	
BEO	*I*	*X*	*?*	-	*I*			*?*	*?*			
BDPO	*I*	*X*		*I*	-					*?*	*?*	
EAR	*I*	*X*	*X*			*I*		*?*	*?*	*?*	*?*	
Melissa	*X*	*?*					*I*					
RYAX	*X*	*?*		*?*		*?*		*I*				*X*
EXAMON	*X*	*?*		*?*		*?*			*I*	*X*	*X*	
CNTD		*?*	*?*		*?*	*?*			*X*	*I*	*?*	
PULP Ctr			*?*		*?*	*?*			*X*	*?*	*I*	
Bebida	*X*	*X*						*X*				*I*

7. Evaluation Plan

The evaluation of REGALE is a highly challenging task. On one hand, REGALE addresses a wide range of strategic objectives with varying quantification/qualification metrics which call for diverse and non-trivial evaluation strategies. We provide below the quantification of these strategic objectives together with their metrics, baselines and targets. On the other hand, evaluation at scale, and ultimately for exascale, is hard, since large-scale operational systems are not easily available for testing system software that needs exclusive and privileged access to the system. To cope with this problem, we will follow an incremental approach, where: a) initial implementations will be tested at laboratory scale using tens to hundreds of processing cores in native or virtualized platforms, b) tested and validated components will be further evaluated on Tier-1 production systems provided by CINECA, BSC and BADW-LRZ, c) we will utilize the PRACE facilities to request for even more resources at Tier-0 systems and d) novel scheduling policies will be evaluated using realistic resource management simulation, directly interfacing production resource managers with Batsim a state-of-art HPC simulator.

STRATEGIC OBJECTIVE 1 (SO1): EFFECTIVE UTILIZATION OF RESOURCES

SO1.1: Improved application performance. By equipping critical applications with computational resources in a sophisticated manner, we will drastically increase their performance (metric: time to solution) compared to current state-of-practice (baseline: execution in a state-of-the-art supercomputer with the current setup). For traditional HPC applications, we expect that this benefit may reach up to a target of 20%. For complex, workflow-based applications, the benefits can reach a target of more than 2x, depending on their resource footprint and data dependencies.

SO1.2: Increased system throughput. Focusing on the system side, by incorporating REGALE innovation in supercomputer operation we aim for an increase in system throughput (metric: committed applications per day) compared to current state of the art (baseline: as above) by at least 30% depending on the mixture of applications to execute on the system.

SO1.3: Minimization of performance degradation under the operation with power constraints. Current practices for power capping are mostly brute-force, i.e. they completely shut down nodes in the supercomputer to enforce the requested level of power consumption. This means that for example 20% in power consumption reduction translates into 20% decrease in system throughput. In REGALE we will consider the resource demands of the application mix under execution to enable effective power capping. Our goal in this case is to at least halve the impact of power capping on throughput (metric: as above) compared to current state-of-the-art (baseline: as above)

SO1.4: Decreased energy to solution. REGALE will be able to support supercomputer operation prioritizing reduced energy consumption. Our goal under this scenario is to achieve at least 15% energy reduction maintaining the same throughput of current state of practice (target: 10%, metric: energy reduction with the same throughput, baseline: as above).

STRATEGIC OBJECTIVE 2 (SO2): BROAD APPLICABILITY

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

Scalability: The REGALE system is targeted to be able to operate in exascale setups and beyond. To assess this objective we will perform experimental results and simulation as described above and we will also extrapolate our results to higher scales. Our goal in this case will be for our prototype system to have minimal overheads at 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

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 to 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 easily interface with the REGALE architecture and system to develop and deploy their code. 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.

8. Conclusions and Future Directions

In this initial deliverable, we sorted out the architecture requirements, the components/interfaces definition, the initial integration plan, 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 initial REGALE architecture, while following the requirements. We then assessed our software tools to realize the use cases based on the architecture and introduced the initial integration and evaluation plans. This deliverable is going to be the milestone for the actual software integration/implementation in WP3 as well as the pointer to missing pieces to be investigated as scientific studies in WP2.

In the future deliverables, 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.

Last but not least, 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.

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] Netti, Alessio, et al. "Dcdb wintermute: Enabling online and holistic operational data analytics on hpc systems." *Proceedings of the 29th International Symposium on High-Performance Parallel and Distributed Computing (HPDC)*. 2020.