

FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

Architecture for the LISA Global Fit Distributed Computing

Diogo André Pereira Babo



Mestrado em Engenharia Informática e Computação

Supervisor: Paulo Jorge Valente Garcia

Second Supervisor: Ademar Manuel Teixeira de Aguiar

July 22, 2025

Architecture for the LISA Global Fit Distributed Computing

Diogo André Pereira Babo

Mestrado em Engenharia Informática e Computação

July 22, 2025

Abstract

The Laser Interferometer Space Antenna (LISA) mission represents one of the most ambitious gravitational wave detection projects, requiring unprecedented computational capabilities for processing heavy workloads through GlobalFit algorithms. This thesis investigates the deployment of the LISA GlobalFit distributed computing architecture on Portugal’s national High-Performance Computing (HPC) infrastructure, specifically the Deucalion supercomputer, to enable Portugal’s participation in the LISA Distributed Data Processing Centre (DDPC) network and create a Portuguese Distributed Computing Center (DCC).

The research addresses the critical challenge and comprehensive analysis of integrating cloud-native scientific workflow orchestration systems with traditional HPC environments and the integration challenges between Singularity-based HPC environments and Kubernetes-based cloud-native architectures.

The LISA GlobalFit framework employs sophisticated Markov Chain Monte Carlo (MCMC) algorithms that require distributed computing across multiple international HPC centers, presenting unique architectural and technological compatibility challenges. The methodology adopted a phased approach, beginning with successfully validating the isolated GlobalFit framework on Deucalion’s heterogeneous ARM-based architecture, followed by attempts to deploy the DCC architecture.

Key findings demonstrate that Portugal has the technical capabilities to contribute meaningfully to LISA data processing. The isolated GlobalFit framework was successfully deployed, achieving 100% CPU utilization and validating critical checkpoint and resume functionality essential for long-running HPC computations. The framework successfully scaled from simple test cases involving single Gaussian fitting to complex scenarios with thousands of sources and samples, confirming its suitability for LISA-scale data processing requirements. However, the research revealed fundamental architectural incompatibilities between traditional HPC containerization technologies (Singularity) and modern cloud-native orchestration requirements (Kubernetes-based systems). The absence of OCI-compatible container engines on Deucalion created insurmountable barriers for deploying the DCC architecture, highlighting broader tensions between traditional HPC infrastructure and evolving cloud-native scientific computing paradigms.

The main contributions are: (1) a replicable methodology for deploying the GlobalFit framework on heterogeneous HPC systems; (2) documentation and architectural analysis filling key gaps in existing resources; (3) a technical assessment of challenges and solutions in multi-architecture deployment; and (4) strategic insights for the EuroHPC community on integrating traditional HPC with cloud-native scientific workflows.

The research establishes Portugal’s technical readiness for LISA participation while identifying critical infrastructure requirements for full DDPC integration. Although DCC architecture deployment was not achieved due to container engine incompatibilities, the work provides valuable insights that can inform future European HPC infrastructure decisions and contribute to the broader understanding of HPC-cloud integration challenges in scientific computing.

UN Sustainable Development Goals

This chapter comprehensively analyzes how this master’s dissertation project aligns with the United Nations Sustainable Development Goals (SDGs) outlined in the 2030 Agenda for Sustainable Development. The work undertaken in developing the architecture for LISA Global Fit distributed computing and deploying it on Portugal’s Deucalion supercomputer contributes meaningfully to several SDGs while advancing scientific knowledge and international collaboration in gravitational wave astronomy.

The analysis follows the framework established by the United Nations for evaluating research projects against the SDGs. This assessment examines how the dissertation’s research objectives, methodology, and outcomes align with specific SDG targets, considering both direct and indirect contributions to sustainable development. The evaluation focuses on identifying the most relevant SDGs, analyzing specific targets within those goals, assessing the project’s contribution to improving scientific capabilities and international cooperation, and establishing performance indicators to measure the project’s impact.

Through systematic analysis of the dissertation’s scope, objectives, and outcomes, three primary SDGs have been identified as directly relevant to this work:

SDG 4: Quality Education - The project significantly advances scientific education and research capabilities by developing and documenting sophisticated computational frameworks for gravitational wave astronomy.

SDG 9: Industry, Innovation and Infrastructure - The work addresses this goal by developing resilient computing infrastructure, promoting innovation in distributed scientific computing, and advancing Portugal’s technological capabilities in high-performance computing.

SDG 17: Partnerships for the Goals - The project exemplifies international scientific collaboration through Portugal’s integration into the LISA Distributed Data Processing Centre network and participation in the broader European high-performance computing ecosystem.

SDG	Goal	Specific Contribution	Performance Indicators and Metrics
4	Quality Education	Development of comprehensive technical documentation and educational resources for distributed scientific computing	• Creation of previously unavailable technical documentation • Development of replicable deployment methodologies • Contribution to scientific education through systematic analysis
9	Industry, Innovation and Infrastructure	Deployment of advanced distributed computing infrastructure and development of innovative integration solutions	• 100% CPU utilization achieved on Deucalion deployment • Successful scaling from simple to complex computational scenarios • Technical readiness demonstrated for LISA participation • Innovative solutions developed for HPC-cloud integration
17	Partnerships for the Goals	Integration into international scientific collaboration networks and strengthening of multi-stakeholder partnerships	• Successful integration into the LISA consortium • Establishment of collaborative relationships within the EuroHPC community • Demonstration of effective public-international partnership • Knowledge sharing through technical contributions

Table 1: Performance Indicators and Metrics aligned with the Sustainable Development Goals

Acknowledgements

I want to express my deepest gratitude to those who have supported me throughout the journey of completing this Thesis. First and foremost, I am sincerely thankful to my supervisor, Paulo Garcia, whose guidance, expertise, and unwavering support were invaluable at every stage of this work. Your encouragement and insightful feedback pushed me to do my best and helped me grow as a researcher. Thank you so much for always caring and helping throughout this Thesis.

I am also very grateful to Tiago Gomes from the thesis team, who was always willing to help, often going above and beyond, even outside regular hours. Your dedication and readiness to assist significantly impacted my research experience.

Finally, I would like to thank my family. While not directly involved in this project, their constant support, encouragement, and presence in my life have provided a strong foundation that enabled me to pursue and achieve my goals.

To all of you, thank you for making this journey possible.

Diogo André Pereira Babo

“Learn from yesterday, live for today, hope for tomorrow. The important thing is not to stop questioning.”

Albert Einstein

Contents

1	Introduction	1
1.1	Motivation	1
1.2	Problem	2
1.3	Research questions	2
2	Background	4
2.1	LISA Mission Overview	4
2.2	Data Analysis Challenges	5
2.3	GlobalFit Algorithm	5
2.3.1	Markov Chain Monte Carlo	6
2.3.2	Distributed Data Processing Centres	7
2.3.3	High-Performance Computing	8
2.4	Containerization and Orchestration	10
2.4.1	Docker	10
2.4.2	Podman	10
2.4.3	Kubernetes	11
2.4.4	Comparison of Docker and Kubernetes	12
2.4.5	Use Cases	12
2.5	Containerization and Orchestration in HPC Environments	12
2.5.1	Singularity Container Platform	13
2.5.2	Open Container Initiative Standards	13
2.5.3	Singularity-CRI and Kubernetes Integration	14
2.5.4	Comparative Analysis of HPC Container Technologies	14
3	State of the Art	16
3.1	Methodology	16
3.2	Previous ESA missions	17
3.2.1	Computational Challenges in GAIA	17
3.2.2	Solutions found in GAIA Data challenges	17
3.2.3	Comparison between GAIA and LISA	18
3.2.4	Computational Challenges in EUCLID	18
3.2.5	Solutions found in EUCLID Data challenges	18
3.2.6	Comparison between EUCLID and LISA	19
3.2.7	Conclusion of the comparison	19
3.3	Distributed Computing for Large-Scale Data Analysis	20
3.3.1	Computational Challenges in LISA’s GlobalFit	20
3.3.2	Current State of the GlobalFit Framework	21
3.4	High-Performance Computing Architectures for LISA	21

3.4.1	Current DDPC architecture	21
3.4.2	Distributed Data Processing Centre Network Implementations	24
3.4.3	Load Balancing and Resource Allocation Strategies for Astrophysics Workloads	24
3.4.4	Data Management and Transfer Protocols for Large-Scale Datasets	24
3.4.5	HPC in the GlobalFit Context	25
3.4.6	Specifics in Deploying GlobalFit-like architecture to a DDPC-like environment	25
3.5	Container Technologies in HPC Environments	26
3.5.1	Integration of Docker and Kubernetes in Traditional HPC Setups	26
3.5.2	Performance Optimization of Containerized Applications on Supercomputers	27
3.5.3	Orchestration Systems for Distributed Scientific Workflows	27
3.5.4	Security Considerations for Containers in High-Performance Environments	28
3.5.5	Comparisons Between Traditional Supercomputing Environments and Modern Container-Based Solutions	28
3.6	Research Gaps and Challenges Identified	29
4	Globalfit Framework	30
4.1	Globalfit Framework	31
4.1.1	Core Concepts of the APC	31
4.1.2	Communication Architecture	33
4.1.3	Component Roles and Responsibilities	35
4.1.4	Engine and Modules	38
4.2	Local Deployment	39
4.3	Challenges	39
4.3.1	Lessons learned	39
5	Deucalion Supercomputer	41
5.1	EuroHPC Initiative and Deucalion's Role	42
5.2	System Architecture and Hardware Specifications	43
5.2.1	Partition Selection and Rationale	45
5.3	Software Environment and Resource Management	45
5.3.1	Containerization	46
5.4	Globalfit framework in Deucalion	46
5.4.1	Implementation Strategy and Methodology	47
5.4.2	Installation and Dependency Management	47
5.4.3	Challenges and Solutions	48
5.4.4	Performance Validation	48
5.5	Technical conclusion	48
6	Deploying the DCC on Deucalion	50
6.1	Current DDPC Architecture Requirements	50
6.2	Deucalion Infrastructure Analysis	51
6.3	Prerequisites, Documentation, and Deployment Tool Analysis	52
6.4	Attempted Solutions	54
6.5	Proposed Solutions and Recommendations	55
6.5.1	Presentation to Deucalion Team	57

7 Conclusion	58
7.1 Summary of Findings	58
7.1.1 Infrastructure Analysis and Containerization Challenges	59
7.1.2 DDPC Deployment Challenges and Architectural Incompatibilities	59
7.2 Contributions	60
7.3 Limitations	61
7.4 Future Work	62
7.5 Concluding Remarks	63
7.6 Alignment with UN Sustainable Development Goals	63
References	65
A APC Globalfit Architecture description	72
B Code Listing: First script	76
C Code Listing: Slurm batch script	79
D Install, run and test the GlobalFit framework Locally	83
D.1 Local Deployment	83
D.1.1 How to install the framework	83
D.1.2 First Run of the APC	83
D.1.3 Second Run of the APC	86
E Globalfit framework in Deucalion	87
E.1 Access and Install dependencies	87
E.1.1 Installation of the Framework in Deucalion	87
E.1.2 Download the binaries	89
E.2 Run the framework on Deucalion	89

List of Figures

2.1	Example of MBHB block output result [23]	6
2.2	Illustration of Markov Chain Monte Carlo method [44]	7
2.3	DDPC Global Architecture [63]	8
3.1	MainDcc Architecture [9]	22
3.2	DCC Architecture [9]	22
3.3	DCC Local Processing [63]	23
3.4	NERSC Shifter workflow [53].	27
4.1	Globalfit APC class diagram.	32
4.2	Communication Architecture diagram.	34
4.3	Component Roles and Responsibilities diagram.	36
4.4	Engine and Modules connection in the GlobalFit	38
5.1	Deucalion supercomputer.	42
5.2	Distribution of peak computational performance across Deucalion's three main partitions, showing the heterogeneous architecture of Portugal's flagship supercomputer [57].	43
5.3	Simplified architecture of the Deucalion supercomputer. ARM and x86/GPU compute nodes (left/right) connect via high-speed InfiniBand to a Lustre parallel file system (center). Login and I/O traffic use Ethernet and NFS. [57].	44
5.4	DCC Architecture [9]	46
6.1	DDPC Global Architecture [63]	50
6.2	DCC Cloud Architecture	51
6.3	DCC Cloud Architecture German Solution	53
D.1	CPU utilization during the first run.	84
D.2	Logs of the first run.	85
E.1	Accessing a compute node	88
E.2	Submitting a Slurm batch script	89
E.3	Logs of the execution of the Slurm script. Confirming the successful run.	90

List of Tables

1	Performance Indicators and Metrics aligned with the Sustainable Development Goals	iii
2.1	Key Differences between Docker and Kubernetes [6, 37, 40]	12
6.1	Proposed Solutions for DDPC Integration with Deucalion	55

Abbreviations and Symbols

AEI	Albert Einstein Institute
APC	Advanced Process Control
CPU	Central Processing Unit
CRI	Container Runtime Interface
DCC	Distributed Computing Center
DDPC	Distributed Data Processing Centres
GPU	Graphics Processing Unit
HPC	High-Performance Computers
HTC	high-throughput computing
KIND	Kubernetes in Docker
LISA	Laser Interferometer Space Antenna
MBHB	Massive Black Hole Binary
MCMC	Markov Chain Monte Carlo
MPI	Message Passing Interface
NCC	National Competence Centre
OCI	Open Container Initiative
SDG	United Nations Sustainable Development Goals
SIF	Singularity Image Format
Slurm	Simple Linux Utility for Resource Management
SSH	Secure Shell

Chapter 1

Introduction

1.1 Motivation

Laser Interferometer Space Antenna (LISA) is a space-based gravitational wave observatory that is responsible for observing the gravitational-wave signals from a variety of systems, ranging from compact stellar objects to supermassive black holes [12]. Data analysis challenges LISA poses arise from the overwhelmingly large number of astrophysical sources that require a simultaneous global fit of a data model [55]. In the context of LISA, these fits are called GlobalFits and usually consist of different types of Markov Chain Monte Carlo (MCMC) algorithms that are so complex (hundreds of parallel branches) and have a long run time (weeks) that they cannot safely be run in standard computing center queues [62]. These algorithms must disentangle overlapping signals from galactic binaries, massive black hole mergers, and extreme mass ratio inspirals while accounting for complex instrumental noise, creating computational requirements that exceed the capabilities of any single computing center [55].

The scale of this challenge has driven the European scientific community to develop the Distributed Data Processing Centre (DDPC) network, a groundbreaking collaboration that links high-performance computing resources across multiple countries to process LISA data [48] collectively. The DDPC network signifies a transformative shift in structuring international scientific collaboration by integrating containerized applications and cloud-native orchestration systems in multiple High-Performance Computing (HPC) centers [86]. At the same time, they communicate with the primary Distributed Computing Center (DCC) to share resources. This integration is crucial as it allows these applications to operate seamlessly across diverse and heterogeneous computing environments [48].

This initiative represents a new paradigm in international scientific collaboration, where containerized applications and cloud-native orchestration systems must seamlessly operate across heterogeneous computing environments spanning from Portugal's Deucalion supercomputer [21] to computing centers in Germany, France, and beyond [48, 72].

1.2 Problem

LISA collects scientific observables that are successively processed into high-level data that we will consider our dataset. The GlobalFit algorithm operates on this dataset and will analyze thousands of overlapping gravitational wave sources that persist for months or years, requiring simultaneous fit of thousands of parameters, hence the name GlobalFit algorithms [62]. The computation chain from raw data to the GlobalFit computations is challenging to perform efficiently [47].

Scaling complex computations at the level that LISA requires makes it imperative to use multiple HPC centers within the LISA consortium [48] – The DDPC network. Numerous countries are involved [72], and the DDPC network will include various HPC nodes. Using this computing power efficiently together with an already working version of the algorithm [45], which was not tailored for distributed computing, is a challenge and, ultimately, where the problem lies. Currently, Europe and US GlobalFits rely on implementations where all blocks are tightly coupled, e.g., to communicate through Message Passing Interface (MPI), a very rigid solution [16]. This is not the intended design for the DDPC, as we should not impact or be impacted by other blocks' ongoing development. In addition, the way to orchestrate the DDPC network that will use the GlobalFit in each HPC currently relies on cloud-native orchestration systems. Many HPCs may require adaptation as they are unprepared for these use cases [86].

Portugal's integration into this global computational network presents specific technical and strategic challenges that exemplify broader European scientific computing landscape issues. The deployment of LISA's distributed computing infrastructure requires bridging traditional high-performance computing environments, optimized for large-scale parallel scientific applications, with modern cloud-native orchestration technologies designed for dynamic, containerized workloads [86]. This technological convergence demands solutions that preserve supercomputing environments' computational efficiency and security requirements while enabling the flexibility and fault tolerance essential for international collaborative science [86].

The fundamental challenge lies not merely in the volume of data that LISA will generate, but in the revolutionary approach required to process it. Therefore, it is paramount to research and create new frameworks capable of satisfying these constraints while minimizing integration efforts and leveraging science activities on the labs' side.

1.3 Research questions

To address these fundamental challenges and assess Portugal's capability to contribute meaningfully to this scientific endeavor, this research investigates four critical questions:

1. Can Portugal's national high-performance computing infrastructure effectively support the computational demands of LISA's Global Fit algorithms while maintaining the performance characteristics required for meaningful gravitational wave data analysis?

2. What are the limitations on the capabilities of the GlobalFit algorithm that will constrain the efficiency of distributed computing clusters?
3. What are the fundamental architectural compatibility challenges between traditional HPC containerization technologies and modern cloud-native orchestration systems, and how do these affect the deployment of distributed scientific computing frameworks?
4. In the context of the Portuguese DCC, to what extent do the existing national HPC capabilities, with their diverse properties, support the execution of the GlobalFit framework?

These questions address the immediate technical challenges of deploying LISA's computational infrastructure and compatibility challenges between HPCs and modern cloud-native orchestration systems. The answers will inform Portugal's specific contributions to gravitational wave astronomy and the development of computational strategies for future large-scale scientific collaborations across the European research landscape.

Chapter 2

Background

This chapter provides the essential background needed to understand the technical and scientific context of the LISA Global Fit project and its computational challenges. Here, we will introduce the main topics covered in Chapter 2 and explain their relevance to the overall goals of the thesis.

The chapter begins with a concise overview of the LISA mission itself, outlining its objectives as a pioneering space-based gravitational wave observatory and highlighting the unprecedented data analysis challenges it introduces. This sets the stage for understanding why developing advanced computational frameworks—such as the GlobalFit algorithm—has become indispensable.

Next, the chapter delves into the complexity of LISA’s data analysis requirements. It explains how the mission’s detection of thousands of overlapping gravitational wave sources, each with unique characteristics and durations, necessitates sophisticated algorithms and high-performance computing strategies. By examining the intricacies of the GlobalFit algorithm and its reliance on Markov Chain Monte Carlo methods, the chapter underscores the computational intensity and distributed nature of the problem.

The discussion then transitions to the broader technological ecosystem, introducing the Distributed Data Processing Centres (DDPCs) concept and the critical role of high-performance computing (HPC) in enabling large-scale, collaborative data analysis across international boundaries. The chapter further explores the evolution of containerization and orchestration technologies—such as Docker, Kubernetes, and Singularity—and their impact on the deployment and scalability of scientific workflows within HPC environments.

By the end of this chapter, the reader will have a clear understanding of the scientific motivations, computational challenges, and technological solutions that underpin the LISA Global Fit project. This background is vital for appreciating the architectural decisions, implementation strategies, and integration efforts discussed in the subsequent chapters.

2.1 LISA Mission Overview

LISA is a space-based gravitational wave observatory [22] designed to detect and observe gravitational wave signals from various astrophysical sources, including compact stellar objects and

supermassive black holes [13, 35]. LISA's mission presents significant data analysis challenges due to the vast number of astrophysical sources that require simultaneous global fitting of data models [55].

LISA consists of three spacecraft flying in a triangular formation, separated by millions of kilometers. Each spacecraft contains free-floating test masses and uses laser interferometry to measure minute changes in the distance between the spacecraft, which could indicate the passage of a gravitational wave [13, 49].

2.2 Data Analysis Challenges

The data analysis challenges in LISA stem from the complexity of gravitational wave signals the mission aims to detect and the time it will take to compute. These challenges encompass multiple aspects of signal processing and interpretation [55]. LISA will detect signals from various sources, including galactic binaries, massive black hole mergers, and extreme mass ratio inspirals [7]. Each source type requires different analysis techniques due to its unique signal characteristics. This diversity in signal types necessitates a flexible and robust data analysis framework capable of handling multiple signal morphologies simultaneously.

Many gravitational wave signals that LISA will detect last for months or years. This extended duration requires continuous analysis of incoming data, presenting challenges in data storage, processing, and real-time analysis [55, 93]. Long-duration signals also increase the likelihood of signal overlap, both in frequency and time. This overlap necessitates sophisticated techniques to disentangle and isolate individual signals from the composite data stream [36].

The algorithms used for LISA data analysis, such as the Global Fit algorithm and MCMC methods, are computationally intensive and time-consuming [55]. These algorithms operate on high-dimensional parameter spaces and require vast amounts of data to be processed simultaneously. As a result, they consume significant computational resources and can take considerable time to execute. This computational burden necessitates a distributed approach to data processing, where work is divided among multiple servers or computing nodes.

To address these computational challenges, LISA employs a distributed computing infrastructure called the DDPC network [55]. This network comprises multiple HPC centers distributed across various locations. The DDPC approach allows for efficient load balancing, distributing computational tasks across available resources, and enabling collaborative analysis by international teams [55]. HPC is crucial in LISA's data analysis strategy as HPC systems utilize parallel computing techniques, cluster computing, and GPU acceleration to process data more efficiently [55].

2.3 GlobalFit Algorithm

The GlobalFit algorithm is a comprehensive approach to analyzing LISA data by simultaneously estimating parameters for all detectable sources along with instrument noise parameters [55]. It is

called *global* because it attempts to fit the entire dataset at once rather than analyzing individual sources in isolation.

Key characteristics of GlobalFit algorithms must be simultaneous parameter estimation for multiple source types, incorporation of instrument noise models, handling of signal overlaps and correlations, and iterative refinement of source parameters and noise models.

These algorithms use MCMC algorithms, and they are computationally intensive due to the high-dimensional parameter space and the need to analyze large volumes of data simultaneously. This necessitates the use of advanced statistical techniques and HPC resources. This Global Fit has, as required, the need to be paused, changed, visualized, and restarted at any time due to the long-duration signals.

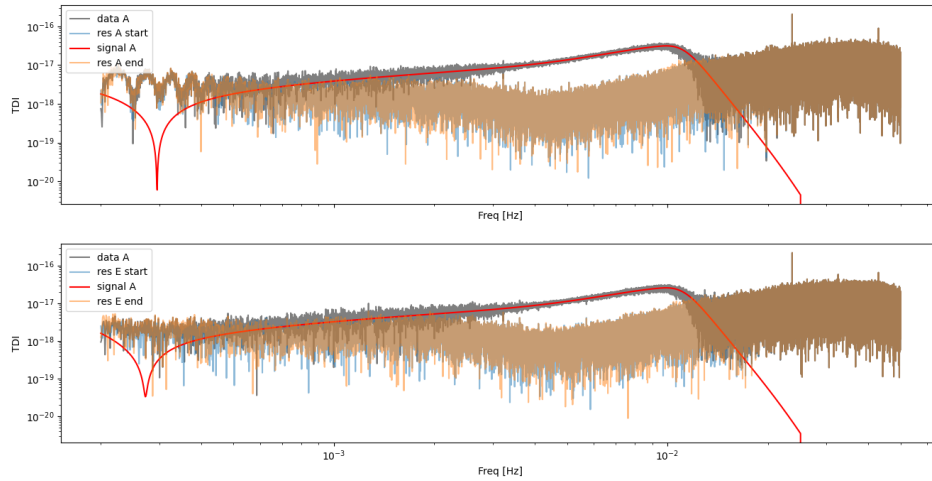


Figure 2.1: Example of MBHB block output result [23]

We can see in Figure 2.1 an example of an output of a massive black hole binary (MBHB) block resulting from an example run of the algorithm.

2.3.1 Markov Chain Monte Carlo

MCMC is a class of algorithms fundamental to LISA's data analysis strategy [20, 88]. MCMC is used for sampling from a probability distribution based on constructing a Markov chain with the desired distribution as its equilibrium distribution [35, 88].

MCMC algorithms are computationally expensive due to the complexity involved and the high dimensionality of the data; when combined with large amounts of data, it poses a computational challenge [88]. There are multiple ways to achieve an MCMC algorithm. For example, the Metropolis-Hastings Algorithm is a standard MCMC method that generates samples with a proposal distribution and an acceptance/rejection step. Another important MCMC technique is Gibbs Sampling, which samples from the conditional distributions of each variable.

MCMC is particularly useful for high-dimensional problems and complex probability distributions [88], making it well-suited for LISA's data analysis challenges. Using a distributed approach to solve this computational hurdle is the solution found by the LISA consortium [48, 87].

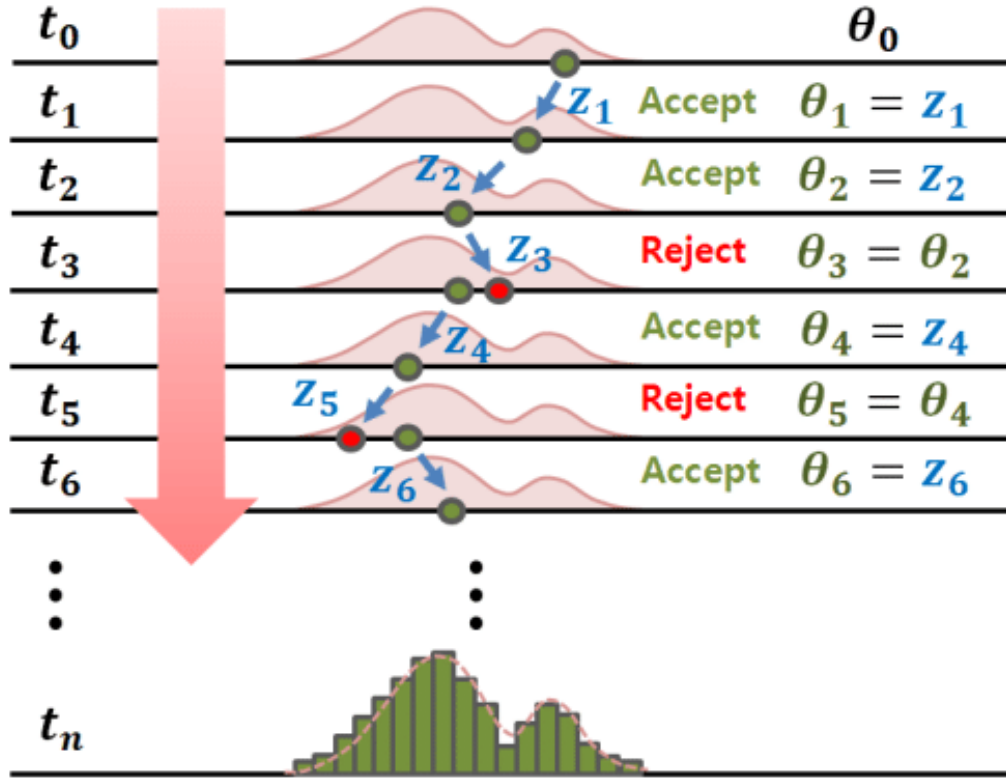


Figure 2.2: Illustration of Markov Chain Monte Carlo method [44]

In Figure 2.2, we can see an example of condition assessments that occur inside an instance of an MCMC method.

2.3.2 Distributed Data Processing Centres

The DDPC network is a crucial component of LISA's data analysis infrastructure [87]. It consists of multiple HPC centers distributed across various countries involved in the LISA consortium [48].

Key features of the DDPC network include distributed computing, which utilizes resources across multiple geographic locations; load balancing to distribute computational tasks across available resources; data redundancy to ensure data integrity and availability through replication; and collaborative analysis capabilities that enable international teams to work on data simultaneously [24].

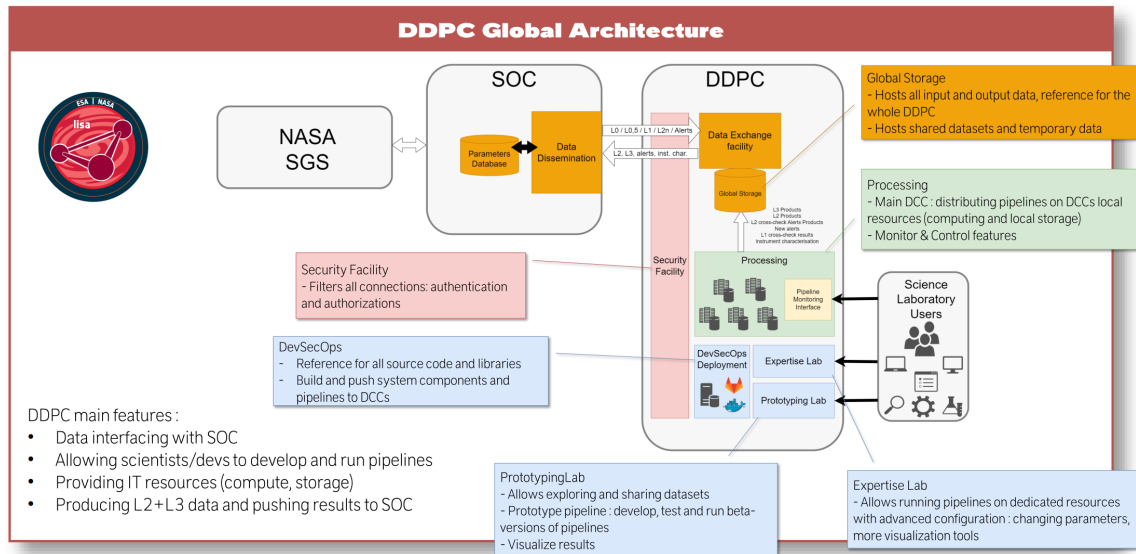


Figure 2.3: DDPD Global Architecture [63]

To better illustrate, we can see in Figure 2.3 a diagram of the DDPD. We will only focus on the DDPD part and go into detail later. Still, the connections made to the outside of the DDPD in the diagram are to the Science Operations Centre (SOC), the central hub where processed scientific data from the LISA mission is delivered and managed for further scientific analysis and operations [63].

2.3.3 High-Performance Computing

HPC refers to using supercomputers and parallel processing techniques to solve complex computational problems [26]. In the context of LISA, HPC is essential due to the enormous computational requirements of the GlobalFit algorithms and the need to process vast amounts of data in real time.

HPC has multiple important key ideas; *parallel computing* is one of them. This involves simultaneously using various computing resources to solve a computational problem. In LISA's case, parallel computing allows for the distribution of complex calculations across numerous processors, significantly reducing the time required for data analysis.

GPU acceleration, also a known technique of HPC, uses graphics processing units to perform computations traditionally handled by the CPU and offers significant speedups for certain types of calculations crucial for LISA's data analysis [48]. For instance, researchers have demonstrated substantial performance improvements in gravitational wave detection using GPU-accelerated computing compared to traditional CPU-based approaches [71].

Cluster Computing is also a form of parallel computing where multiple computers work together as a single system. For LISA, cluster computing enables the creation of robust, scalable systems capable of handling the mission's data processing demands.

2.3.3.1 Slurm Workload Manager

A critical component of modern HPC environments is the workload manager, which orchestrates job scheduling and resource allocation across computing clusters [8, 75]. The Slurm Workload Manager, known initially as Simple Linux Utility for Resource Management, represents one of the most widely adopted solutions in this domain [94]. Developed initially at Lawrence Livermore National Laboratory in 2002 by Morris Jette and Mark Grondona, Slurm was created to address the lack of simple, highly scalable, and portable resource managers available for Linux clusters [64, 94].

Slurm operates as an open-source, fault-tolerant, and highly scalable cluster management and job scheduling system designed for Linux clusters ranging from small installations to systems with thousands of nodes [73, 94]. The system provides three fundamental functions that are essential for HPC operations: allocating exclusive and non-exclusive access to compute nodes for users to perform computational work, providing a comprehensive framework for starting, executing, and monitoring parallel jobs on allocated resources, and managing contention for resources through sophisticated queue management of pending jobs [75, 91].

The architecture of Slurm centers around a lightweight, modular daemon structure consisting of two primary components [83]. The central controller daemon, `slurmctld`, runs on management nodes and handles scheduling decisions, job queue management, and overall state tracking of the cluster [91]. On each compute node, a worker daemon called `slurmd` executes local operations including task launching, resource monitoring, and status reporting back to the central controller [59, 91]. This design ensures fault-tolerant hierarchical communications while avoiding reliance on less secure methods such as SSH for job execution [91].

Slurm's scheduling capabilities utilize sophisticated algorithms to optimize resource utilization and minimize job waiting times [59]. The system employs a best-fit algorithm based on Hilbert curve scheduling or fat tree network topology to optimize locality of task assignments on parallel computers [75]. Modern implementations incorporate priority-based scheduling where jobs age over time, and advanced backfill scheduling algorithms that attempt to schedule lower-priority jobs without delaying higher-priority ones [38, 59]. The remarkable success of Slurm in the HPC community is evidenced by its adoption on approximately 60% of the TOP500 supercomputers, making it the *de facto* standard for workload management in high-performance computing environments [73, 75].

2.3.3.2 Portuguese National Cluster

Portugal has been making significant strides in HPC platforms. It actively participates in the European HPC landscape, crucial for supporting ambitious scientific endeavors like LISA [85].

The Portuguese National Competence Centre (NCC) for HPC was established as part of the EuroCC project, which aims to create a network of national competence centers across Europe. This center plays a vital role in enhancing and developing the competencies of the Portuguese

computational community, with a focus on making full use of EuroHPC resources and partnerships [85].

A significant milestone in Portugal's HPC capabilities was the inauguration of the Deucalion supercomputer in 2023 [85]. It's also important to note that time in this supercomputer is free of charge to researchers who submit accepted proposals. We will dive into the specific details of Deucalion later, as this is the HPC center we will use.

2.4 Containerization and Orchestration

Since the solutions for the LISA DDPC will be using Containerization and Orchestration, as we will explain later in the state of the art, there is a need to explain the concepts and the two most well-known Containerization and Orchestration technologies.

Docker and Kubernetes are complementary technologies that have revolutionized modern application development and deployment [14]. While Docker focuses on containerization, Kubernetes excels in container orchestration.

2.4.1 Docker

Docker is a platform for developing, shipping, and running container applications. Containers are lightweight, standalone, and executable packages that include everything needed to run an application: code, runtime, system tools, libraries, and settings [40].

Docker's strength lies in its ability to create industry-standard containers, ensuring portability across different environments [40]. These lightweight containers share the host OS kernel, which reduces overhead and improves efficiency. Docker provides isolation for applications, enhancing security and reducing conflicts. Its containers can run consistently across various computing environments, from development to production. Additionally, Docker's resource efficiency allows it to handle more applications with fewer resources compared to traditional virtual machines [40].

2.4.2 Podman

Podman, short for "pod manager," represents a significant evolution in container technology that addresses several limitations inherent in Docker's architecture [41, 90]. Developed by Red Hat engineers in collaboration with the open-source community, Podman is an open-source tool designed to create, manage, and run containers while maintaining full compatibility with Docker's command-line interface [41, 61]. The fundamental architectural difference between Podman and Docker lies in Podman's daemonless design, which eliminates the need for a persistent background service with root privileges [4, 70].

The daemonless architecture of Podman provides several critical advantages over traditional container engines [70, 90]. Unlike Docker, which relies on a centralized daemon process that requires root privileges to manage container lifecycle operations, Podman executes containers directly as child processes of the user who invoked them [4, 41]. This design eliminates the single

point of failure that Docker's daemon architecture introduces, where a daemon crash can terminate all managed containers [70]. Additionally, Podman's approach significantly enhances security by allowing containers to run without root access, making it particularly suitable for multi-user environments such as HPC clusters [41, 78].

Podman's rootless container capabilities represent a paradigm shift in container security and usability [41, 78]. When operating in rootless mode, Podman allows regular users to deploy and manage containers without elevated privileges, utilizing Linux user namespaces to provide isolation while maintaining the user's identity within the container [78]. This approach ensures that users cannot access resources beyond their permitted scope, effectively preventing privilege escalation attacks in daemon-based systems [43, 70]. The rootless functionality is enabled by default for newly created users and stores all container-related data in the user's home directory rather than system-wide locations [78].

The compatibility between Podman and Docker extends beyond mere command-line interface similarity [42, 90]. Podman can directly pull and run Docker containers without modification, and its adherence to Open Container Initiative standards ensures interoperability across different container ecosystems [61, 90]. Performance comparisons demonstrate that Podman achieves comparable or superior performance to Docker in many scenarios, particularly in registry access and image handling operations [27, 42]. For users transitioning from Docker environments, Podman provides an alias mechanism that allows direct substitution of Docker commands with Podman equivalents, requiring minimal changes to existing workflows [42, 90].

2.4.3 Kubernetes

Kubernetes is an open-source system for automating containerized application deployment, scaling, and management [50]. It builds upon the containerization concept, providing a robust framework for running distributed systems. For the LISA project, Kubernetes is essential as it is the technology used to deploy the GlobalFit architecture that ensures the integrity of the DDPC, which we will see in later chapters.

Kubernetes offers a comprehensive set of features for managing containerized applications at scale. It provides service discovery and load balancing, exposing containers using DNS names or IP addresses and distributing network traffic efficiently. Kubernetes excels in orchestration, automatically mounting storage systems to containers as needed [67]. It manages the desired state of containerized applications, allowing for automated rollouts and rollbacks. The platform's automatic bin-packing capability efficiently allocates containers to nodes based on resource requirements. Kubernetes implements self-healing mechanisms, automatically restarting failed containers and rescheduling containers when nodes die [89]. It also offers robust secret and configuration management, handling sensitive information and application configurations without rebuilding container images. Furthermore, Kubernetes provides horizontal scaling, adjusting application resources based on CPU usage or other defined metrics [67].

2.4.4 Comparison of Docker and Kubernetes

Aspect	Docker	Kubernetes
Scope	Containerizing applications	Orchestrating and managing containerized applications at scale
Functionality	Creates runs, and manages individual containers	Provides a framework for running distributed systems and managing clusters of containers
Scalability	Primarily concerned with individual container operations	Excels at managing large numbers of containers across multiple hosts
Complexity	Relatively simple for individual container management	Steeper learning curve but offers more advanced features for complex deployments

Table 2.1: Key Differences between Docker and Kubernetes [6, 37, 40]

2.4.5 Use Cases

Docker: Docker finds its primary applications in development environments, CI/CD pipelines, microservices architecture, and application packaging and distribution. Its simplicity and efficiency make it ideal for these scenarios where individual container management is key [14].

Podman: Podman is increasingly adopted when enhanced security, rootless operation, and compatibility with existing Docker workflows are critical. Its daemonless and rootless architecture makes it particularly suitable for multi-user environments such as academic clusters and HPC systems, where minimizing privilege escalation risks is essential [27].

Kubernetes: Kubernetes is made for large-scale production deployments, multi-cloud and hybrid cloud environments, complex microservices architectures, high-availability applications, and scenarios requiring auto-scaling and load balancing of containerized applications. Its robust orchestration capabilities make it suitable for managing complex, distributed systems [79].

2.5 Containerization and Orchestration in HPC Environments

Since all components of the LISA DDPC network will be HPC, there is a need to understand how current containerization technologies interact in HPC Environments. Because HPC environments present unique challenges for containerization technologies that differ significantly from traditional cloud and enterprise computing scenarios [18, 25]. The demanding requirements of HPC workloads, including maximum performance, specialized hardware access, and multi-user security, have driven the development of container technologies specifically designed for scientific

computing applications [34, 51]. Unlike general-purpose containerization solutions, HPC containerization must address concerns such as seamless integration with job schedulers, native MPI support, GPU acceleration capabilities, and the ability to operate without compromising system security in shared computing environments [2, 18].

2.5.1 Singularity Container Platform

Singularity represents the most prominent containerization solution designed specifically for HPC environments [18, 80]. Developed initially at Lawrence Berkeley National Laboratory, Singularity was created to address the fundamental incompatibilities between Docker's security model and the requirements of multi-user HPC systems [34, 80]. The platform enables researchers to create and run containers that package software applications in a portable and reproducible manner while maintaining the security and performance characteristics essential for scientific computing [76, 80].

The architectural philosophy of Singularity prioritizes integration over isolation, a design decision that distinguishes it fundamentally from Docker and other cloud-oriented container platforms [18, 80]. Singularity containers execute under the same user identity as the host system, eliminating the privilege escalation risks associated with Docker's root daemon architecture [25, 43]. This design enables seamless access to high-performance networks, parallel filesystems, and specialized hardware such as GPUs without requiring complex configuration or security compromises [18, 80]. The integration approach allows HPC applications to maintain their performance characteristics while benefiting from the portability and reproducibility that containerization provides [34, 51].

Singularity's image format, known as the Singularity Image Format (SIF), provides several advantages for HPC deployments [76, 80]. SIF images are immutable, single-file containers with cryptographic signatures for verifiable reproducibility and security [80]. This design facilitates easy transport and sharing of computational environments across different HPC facilities while ensuring that the containerized applications remain unchanged during transfer [76]. The mobility of SIF containers enables researchers to develop applications on local workstations and deploy them seamlessly on large-scale HPC systems without modification [18, 80].

2.5.2 Open Container Initiative Standards

The Open Container Initiative (OCI) represents a collaborative effort to establish vendor-neutral standards for container formats and runtimes across the computing industry [33, 58]. Founded in June 2015 under the Linux Foundation, the OCI emerged from recognizing that the rapidly growing container ecosystem required standardization to prevent vendor lock-in and ensure portability across different implementations [17, 33]. The initiative combines major technology companies and open-source projects to develop specifications that preserve container technology's flexible and open nature while establishing common standards [33, 58].

The OCI specifications encompass three fundamental areas that define modern container technology [17, 58]. The Runtime Specification defines the configuration, execution environment, and lifecycle management of containers, establishing how container engines should create and manage running containers [17, 33]. The Image Specification standardizes the container image format, enabling interoperability between different tools for building, storing, and distributing container images [17, 33]. The Distribution Specification addresses the protocols and APIs for distributing container images across registries and storage systems [58]. Together, these specifications ensure that any OCI-compliant runtime can execute containers built with one set of tools, providing users with vendor choice and implementation flexibility [17, 33].

The significance of OCI compliance extends beyond mere standardization to encompass the practical needs of diverse computing environments [33, 56]. For HPC applications, OCI standards provide a foundation for developing container solutions that can operate across different supercomputing facilities while maintaining compatibility with existing scientific workflows [56]. The standardization enables HPC centers to adopt container technologies with confidence that their investments will remain viable as the container ecosystem evolves [33].

2.5.3 Singularity-CRI and Kubernetes Integration

Integrating Singularity with Kubernetes orchestration represents a significant advancement in bringing cloud-native workflow management to HPC environments [56]. The Singularity Container Runtime Interface (CRI) enables Kubernetes to orchestrate Singularity containers for compute-intensive workloads, bridging the gap between cloud orchestration technologies and HPC-specific container requirements [56]. This integration requires both an OCI-compliant runtime implementation within Singularity and a specialized CRI that can interface with Kubernetes while maintaining Singularity's HPC-oriented features [56].

The technical implementation of Singularity-CRI involves significant modifications to support OCI compliance while preserving the unique characteristics that make Singularity suitable for HPC applications [56]. Singularity Image Format files encapsulate the necessary components for instantiating OCI-compliant runtimes, enabling compatibility with Kubernetes orchestration while maintaining the security and performance benefits of Singularity's original design [56]. This dual-compliance approach allows HPC workloads to benefit from Kubernetes' sophisticated scheduling and management capabilities without sacrificing the specialized features required for scientific computing applications [56].

2.5.4 Comparative Analysis of HPC Container Technologies

Understanding that the LISA consortium comprises multiple HPCs, which can vary in tools and available resources, it's essential to know how these containerization technologies interact since they will eventually have to run towards a common goal.

The landscape of container technologies in HPC environments reflects the diverse requirements and constraints of scientific computing workloads [2, 5]. While revolutionary for cloud

and enterprise applications, Docker faces significant limitations in HPC contexts due to its security model and architectural assumptions [19, 43]. The requirement for root privileges and the daemon-based architecture create unacceptable security vulnerabilities in multi-user HPC environments where users must be prevented from accessing other users' data or gaining elevated system privileges [19, 43].

Podman addresses some of Docker's limitations through its rootless and daemonless architecture, making it more suitable for certain HPC use cases [2, 27]. However, Podman's design still reflects its origin in cloud and enterprise computing rather than the specialized requirements of scientific workloads [2]. While Podman can operate without root privileges and provides better security characteristics than Docker, it lacks many of the HPC-specific features that Singularity delivers, such as native MPI support and seamless integration with HPC job schedulers [18, 27].

Singularity emerges as the most HPC-appropriate solution due to its design philosophy, which prioritizes the unique requirements of scientific computing environments [18, 51]. The platform's ability to execute containers with the same user identity as the host system, its native support for parallel filesystems and high-performance networks, and its integration with job scheduling systems like Slurm make it the preferred choice for most HPC applications [2, 25]. Performance evaluations consistently demonstrate that Singularity containers achieve near-native performance for HPC workloads, with minimal overhead compared to non-containerized applications [2, 51]. Combined with Singularity's security model and HPC-specific features, this performance characteristic establishes it as the standard containerization solution for scientific computing environments [18, 25].

Chapter 3

State of the Art

This chapter is a comprehensive review of the current advancements and challenges in distributed computing and HPC systems, particularly in the context of the LISA mission. This chapter systematically addresses the critical components and technologies underpinning LISA’s computational framework, ensuring a coherent flow from general concepts to specific applications.

The structure of the chapter is intentionally designed to guide the reader through increasingly specific topics. It begins with comparing LISA’s computational needs with those of previous ESA missions such as Gaia and Euclid, drawing lessons from their approaches to similar challenges. Then we introduce an overview of distributed computing for large-scale data analysis, highlighting computational challenges unique to LISA’s GlobalFit algorithm. This sets the stage for understanding why advanced HPC architectures are necessary. Subsequently, the chapter delves into detailed discussions on HPC solutions tailored for LISA, including DDPCs, containerization technologies like Docker and Kubernetes, and their integration into traditional HPC environments. Finally, we identify research gaps and challenges.

The focus on LISA throughout the chapter is deliberate, as its unique computational demands necessitate novel frameworks that push the boundaries of current HPC and distributed computing technologies.

3.1 Methodology

The specialized scope and novel nature of the LISA project present unique challenges in accessing comprehensive literature, with the most relevant research found in conference proceedings, unpublished reports, or older broad-coverage documents. This limitation stems from the fact that most existing work is either purely physics-related or consists of past space mission documents that are not publicly available.

A systematic approach was employed to collect relevant literature using structured search queries across databases including IEEE Xplore, Scopus, Science Direct, Web of Science, Google Scholar, and NASA/ADS. The search strategy focused on four main query areas: LISA-specific

research, distributed and high-performance computing, Markov Chain Monte Carlo data processing, and containerized supercomputing solutions.

The literature screening process resulted in fewer than 100 articles meeting the eligibility criteria from an initial pool of 1,609 articles, providing insights into distributed data processing and the computational challenges unique to the LISA mission. Quality assessment prioritized Q1 and Q2 papers while maintaining inclusivity for Q3 and Q4 papers due to the limited quantity of available research.

3.2 Previous ESA missions

In this section, we will provide details about ESA's oversight of the Gaia and Euclid space missions and explore their details and differences from those of LISA. Compare the data challenges from these three missions and analyze the solutions found in the LISA context.

3.2.1 Computational Challenges in GAIA

The Gaia mission faced significant computational challenges due to its enormous data collection. Gaia's primary objective was to create a three-dimensional map of the Milky Way, which required processing data from over a billion stars. This resulted in petabytes of raw data that needed to be analyzed and transformed into a usable catalog [60].

One of the main challenges was the iterative nature of the data processing. The Gaia data reduction required multiple passes through the entire dataset to refine astrometric solutions, as each star's position and motion affected the calculations for other stars. This iterative process, known as the Astrometric Global Iterative Solution (AGIS), demanded substantial computational resources and sophisticated algorithms to converge on accurate results [60].

Another challenge was the need for real-time processing of certain data products, such as alerts for transient phenomena. This required a robust and efficient pipeline capable of quickly analyzing incoming data and identifying objects of interest [60].

3.2.2 Solutions found in GAIA Data challenges

To address these challenges, the Gaia mission implemented several innovative solutions. First, they developed a distributed processing architecture allowing the workload to spread across multiple European data centers. This was the GAIA Data Processing and Analysis Consortium (DPAC). This distributed approach enabled parallel processing of the massive dataset and improved overall efficiency as well as divided responsibilities across Coordination Units (CUs) and Data Processing Centers (DPCs), enabling distributed and parallel development [60].

The mission also employed advanced database technologies and optimized data structures to manage the large volume of information. They developed a custom-built, highly efficient database system called the Main Database, which was designed to handle the specific requirements of the Gaia data model [60].

For real-time processing needs, Gaia implemented a dedicated pipeline with optimized algorithms to detect and characterize transient events rapidly. This system could generate alerts within 24-48 hours of observation [60].

3.2.3 Comparison between GAIA and LISA

While both Gaia and LISA are ambitious ESA missions, they differ significantly in their data processing requirements. Gaia primarily dealt with a large volume of relatively homogeneous data (stellar positions and motions). In contrast, LISA must process complex gravitational wave signals from various sources [55].

The iterative nature of Gaia's data processing is somewhat similar to LISA's GlobalFit algorithm, which also requires multiple passes through the data to disentangle overlapping signals. However, LISA's challenge is more complex due to the need to fit various types of gravitational wave sources simultaneously [55].

GAIA and LISA rely heavily on distributed architectures to process vast data volumes efficiently. Although Gaia's distributed processing architecture could serve as a model for LISA, LISA will require even more sophisticated algorithms and possibly more advanced computing technologies like GPU acceleration and other previously discussed techniques to handle its unique data analysis challenges [55, 60].

3.2.4 Computational Challenges in EUCLID

Euclid, another ESA mission aimed at mapping the geometry of the dark universe, faces its computational challenges. The primary challenge for Euclid is the processing and analysis of a vast amount of imaging and spectroscopic data from billions of galaxies. This mission requires processing compressed datasets of 200 TB, which expand to several petabytes post-processing [28, 65].

One major challenge is the need for precise image processing to measure weak gravitational lensing effects. This requires sophisticated algorithms to correct for instrumental effects, atmospheric distortions, and exact shape measurement techniques [28, 65].

Another challenge is the computation of photometric redshifts for billions of objects, which requires handling and integrating data from multiple wavelengths, including ground-based observations. This process is computationally intensive and requires robust machine-learning algorithms [28, 65].

3.2.5 Solutions found in EUCLID Data challenges

Euclid has implemented a distributed processing system similar to Gaia's to address these challenges, with multiple Science Data Centers (SDC) across Europe. They have developed a modular pipeline architecture that allows for flexibility in algorithm development and implementation [28, 65].

For weak lensing measurements, Euclid has invested in developing and testing multiple independent shape measurement algorithms to cross-validate results and minimize systematic errors. They have also implemented sophisticated image simulation tools to calibrate these algorithms [28, 65].

Euclid employs advanced machine learning techniques, including deep learning approaches, to handle the photometric redshift. They are also developing methods to incorporate spectroscopic redshift information to calibrate and improve photometric redshift estimates [28, 65].

Euclid extensively uses containerization technologies like Docker to ensure consistency across different computing environments and to facilitate the integration of algorithms developed by other teams [28, 65].

3.2.6 Comparison between EUCLID and LISA

While Euclid and LISA have different scientific goals, they share similarities in their data processing challenges. Both missions extract weak signals from noisy data – weak lensing for Euclid and gravitational waves for LISA [28, 65].

Euclid’s approach to distributed computing and containerization could provide valuable insights for LISA’s data processing architecture. However, LISA’s data will be fundamentally different, requiring real-time analysis and more complex signal processing techniques [28, 65]. Euclid’s experience in handling and integrating diverse datasets (space-based and ground-based) could be relevant to LISA, but fundamentally differs from LISA as the DDPCs will only focus on LISA as opposed to incorporating information from multiple sources, in this case electromagnetic counterparts to gravitational wave events [28, 65].

3.2.7 Conclusion of the comparison

In conclusion, while Gaia, Euclid, and LISA face different specific challenges, they all share the common theme of processing massive, complex datasets to extract subtle scientific signals. LISA can learn from the distributed computing architectures of both Gaia and Euclid and their experiences in developing modular, flexible pipeline systems.

However, LISA will need to go beyond these approaches due to the unique nature of gravitational wave data analysis. The real-time processing requirements and the need to disentangle multiple overlapping signals will require novel algorithms and possibly more advanced computing technologies.

The experiences of Gaia and Euclid in cross-validating results using multiple independent analysis techniques could be particularly valuable for LISA, given the unprecedented nature of its measurements. Additionally, the emphasis on simulation and mission calibration will likely be crucial for LISA.

Ultimately, while LISA can draw inspiration and learn lessons from these previous missions, it must develop unique solutions tailored to the specific challenges of gravitational wave astronomy.

3.3 Distributed Computing for Large-Scale Data Analysis

The GlobalFit algorithm is a crucial component of the LISA mission, designed to simultaneously analyze and extract gravitational wave signals from millions of overlapping sources immersed in complex, time-varying instrument noise [55]. This algorithm represents a significant challenge in astrophysical data analysis due to its inherent complexity, involving high-dimensional parameter spaces, long-duration computations often spanning weeks, and the need to disentangle superposed signals from various astrophysical sources [9].

The GlobalFit approach is necessary because LISA’s data stream will contain signals from many sources, including galactic binaries, massive black hole mergers, and extreme mass ratio inspirals, which may overlap in time and frequency [46]. This overlap, combined with the long duration of many signals, necessitates a comprehensive analysis that considers all sources simultaneously to avoid biases and source confusion [54].

3.3.1 Computational Challenges in LISA’s GlobalFit

The computational demands of the GlobalFit algorithm are high, primarily due to the thousands of objects that need to be fit at the exact moment, which translates to thousands of parameters on the MCMC. These challenges are further compounded by the need for scalability across DDPCs, essential for handling the massive computational load [47, 55].

One of the primary issues facing the GlobalFit implementation is that it is currently incompatible with existing supercomputing queue systems and needs to be adapted. Traditional HPC environments typically operate on job scheduling systems with predefined time limits [82]. However, the GlobalFit algorithm’s long-running nature and dynamic resource requirements make it challenging to fit within these constraints.

The algorithm’s complexity also leads to issues with resource locking, where computational resources may be tied up for extended periods, potentially impacting the efficiency of the overall data processing pipeline. This situation is particularly problematic in shared HPC environments where resource allocation must be balanced across multiple users and projects [46].

Current implementations of the GlobalFit algorithm have significantly addressed some of the previously discussed challenges. Notable examples include:

1. **Erebor:** A GPU-accelerated multi-source global fit pipeline capable of analyzing current state-of-the-art datasets [46]. This pipeline has successfully recovered posterior distributions for all injected Massive Black Hole Binaries (MBHBs) and cataloged thousands of Galactic Binaries in the LISA Data Challenge 2A datasets [46].
2. **GLASS:** Global LISA Analysis Software Suite: A modular pipeline using a blocked Metropolis-Hastings algorithm to alternately fit populations of ultra-compact galactic binaries, known verification binaries, massive black hole mergers, and instrument noise models [46, 55].
3. **Time-evolving weekly analysis:** A novel approach that performs global fit analysis in weekly increments, starting from one week of observation time up to a full year [46, 77].

This method allows for detecting more galactic binaries and massive black hole binaries as observation time increases.

Despite these advancements, current implementations face limitations in distributed environments. Many existing solutions rely on MPI for inter-process communication, which can be rigid and poorly suited for the dynamic nature of the GlobalFit problem. The tight coupling between different algorithm components also poses challenges for distributed execution, as it can lead to increased communication overhead and reduced fault tolerance [82].

3.3.2 Current State of the GlobalFit Framework

Efforts are ongoing to develop distributed frameworks that can better handle task synchronization and fault tolerance in the context of the GlobalFit algorithm.

The current framework has several key features that enhance its functionality and usability. Its modular architecture allows different labs to develop components using various languages and technology stacks, facilitating the integration of specialized algorithms and fostering collaboration among research groups [16]. To address the long duration of GlobalFit computations, the framework incorporates a checkpoint/resume feature, enabling jobs to be paused, saved, and resumed. This is an essential capability for adapting to the constraints of HPC queue systems and ensuring fault tolerance [16]. It supports iterative processing with synchronization points between iterations, accommodating a varying number of estimators at each step to dynamically adapt to the evolving needs of the analysis [16]. The architecture is scalable across multiple Distributed Computing Centers (DCCs) and adapts to various available resources, meeting the massive computational demands of LISA data analysis.

3.4 High-Performance Computing Architectures for LISA

3.4.1 Current DDPC architecture

To understand the DDPC architecture, we will explain a single DCC and then expand the view. The Architecture of each DCC is present in Figure 3.2. Expanding the view, we need to know how they communicate and are orchestrated, represented in Figure 3.1. Together, these two Figures provide an overview of what the DDPC comprises.

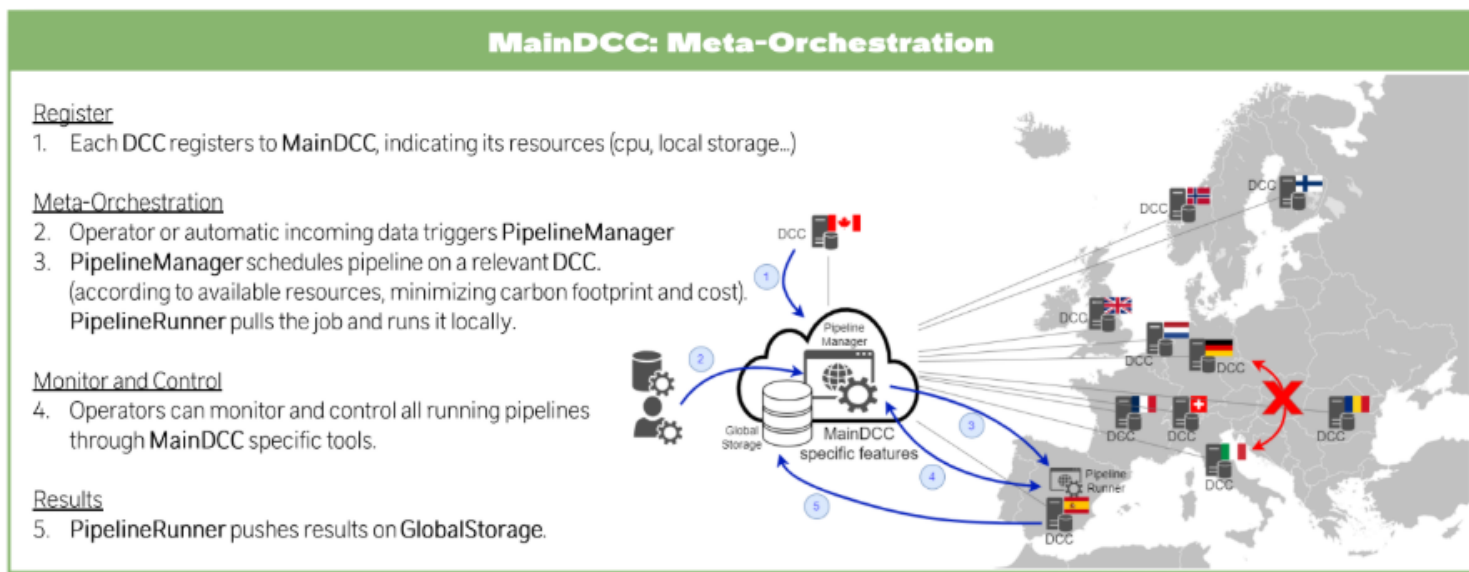


Figure 3.1: MainDcc Architecture [9]

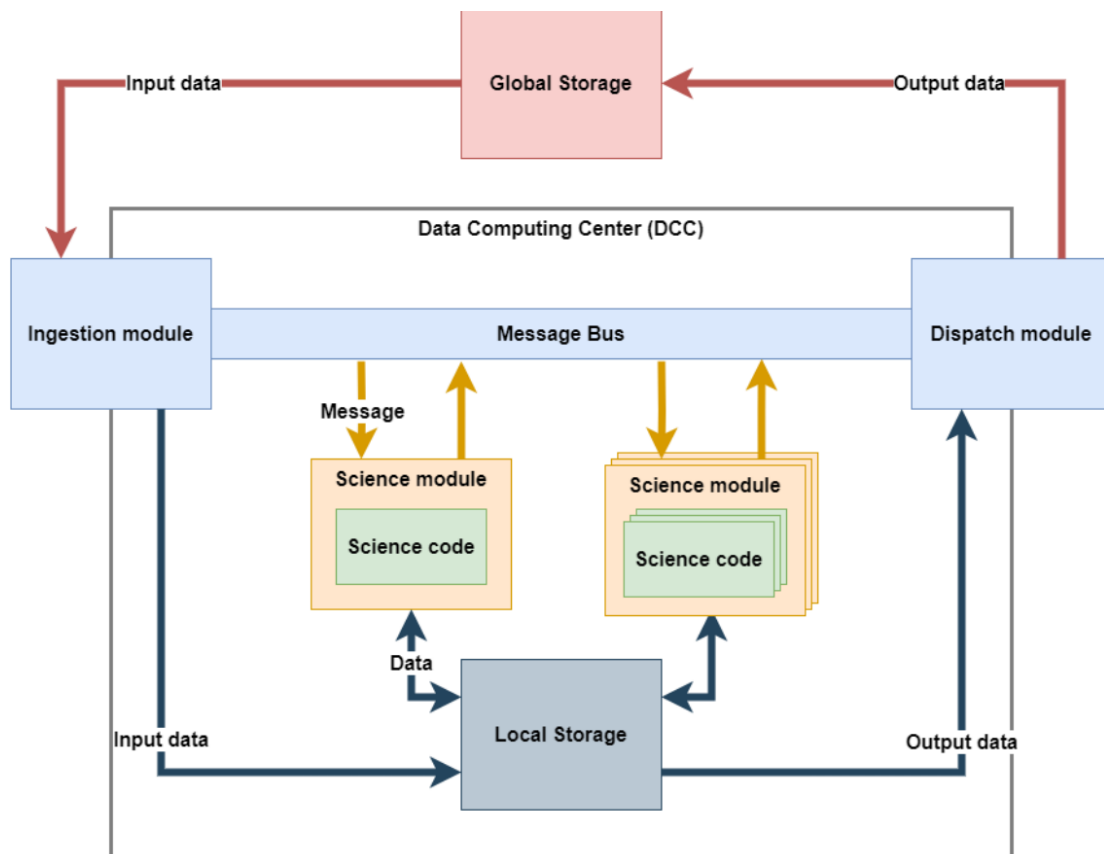


Figure 3.2: DCC Architecture [9]

The DDPC network consists of:

- **MainDCC:** Acts as a meta-orchestrator, managing the registration of DCCs and scheduling pipelines across the distributed infrastructure [63].
- **Pipeline Manager:** Responsible for scheduling pipelines on relevant DCCs based on available resources, carbon footprint, and cost considerations [63].
- **Pipeline Runner:** Executes the scheduled jobs locally within each DCC [63].
- **Global Storage:** Centralizes input and output products, facilitating data sharing across the distributed environment [9].
- **Message Bus:** Enables communication between different components of the framework [9].
- **Modules:** Encapsulate specific algorithmic components, such as noise or gravitational wave source estimators [9].

The framework leverages containerization technologies, particularly Kubernetes, to ensure portability across different DCC environments [63]. This abstraction layer helps implement high availability and consistently perform upgrades across the distributed infrastructure. We can also better illustrate and explain the technology stack using Figure 3.3.

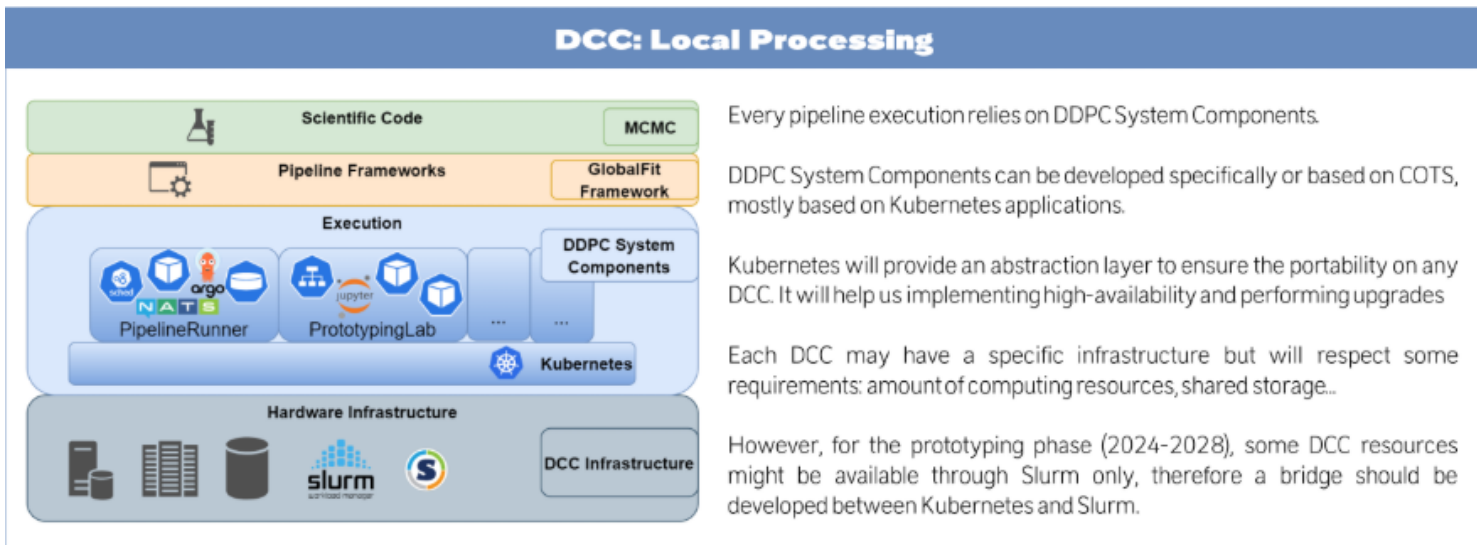


Figure 3.3: DCC Local Processing [63]

For the prototyping phase (2024-2028), the framework is designed to bridge Kubernetes and Slurm job scheduling systems, accommodating DCCs that may only have Slurm-based resources available [63] as we can verify in Figure 3.3.

These three Figures, 3.1, 3.2, 3.3, provide a very detailed illustration of what the DDPC network consists of and how each DCC communicates. Given that multiple components responsible for the DDPC are highlighted in these three figures, our focus will be only on the DDPC part and how to deploy the specific part of the GlobalFit. Hence, we did not provide more details about

all components; we just highlighted the research focus and the components necessary for further deployment, as we will see ahead.

3.4.2 Distributed Data Processing Centre Network Implementations

This network ensures scalability, fault tolerance, and resource sharing by distributing computational workloads across geographically dispersed nodes. Each DCC operates semi-independently, with centralized coordination through the MainDCC for task orchestration and resource allocation. The DDPC relies heavily on containerization technologies like Kubernetes to ensure portability and uniformity across diverse infrastructures, including Slurm-based systems during the prototyping phase [63, 66].

One of the major problems lies in integrating all HPCs with the same type of architecture. Even when using containerization technologies, research is still needed to achieve a "one for all" solution because all HPCs are unique and have different architectures. The case worsens when involving exceptional security and specific HPC requirements because the requirements and protocols are different for each [84].

3.4.3 Load Balancing and Resource Allocation Strategies for Astrophysics Workloads

Efficient load balancing is critical for LISA's GlobalFit algorithm due to its high computational demands and dynamic resource requirements. Strategies include:

1. *Dynamic Resource Allocation:* Using adaptive algorithms like AMS-ERA, which optimize CPU, memory, and disk usage based on job profiles to enhance performance in heterogeneous clusters [68].
2. *Quantum-Assisted Load Balancing:* Emerging approaches like quantum annealing can optimize task distribution by minimizing inter-processor communication and addressing bottlenecks in parallel computations [69].
3. *Fine-Grained Resource Management:* Techniques such as dynamically adjusting hardware priorities in multi-threaded processors to reduce imbalances in parallel applications [15].

Since it will be run in a DDPC environment, the load-balancing techniques should not be called in a DCC but scaled to accommodate all DCCs. This means the framework must address Resource Allocation, Load Balancing, and Resource Management for a whole network of computers. This area also needs further research to achieve the best solution for implementing the GlobalFit Framework.

3.4.4 Data Management and Transfer Protocols for Large-Scale Datasets

LISA's data management framework must address the challenges of handling vast datasets generated by gravitational wave observations. Key elements include:

1. *Centralized Global Storage*: Facilitates data sharing and redundancy across the DDPC network while ensuring consistency and availability [9].
2. *High-Bandwidth Networking*: As the DCCs will be part of a network called GÉANT that already features NUMA-like architectures for efficient data exchange between nodes [1], it is essential to utilize these resources to achieve large-scale model training and analysis [31].
3. *FAIR Data Principles*: Ensuring data findability, accessibility, interoperability, and reusability through standardized formats and metadata systems [10].

3.4.5 HPC in the GlobalFit Context

The GlobalFit algorithm is central to LISA’s mission, requiring robust HPC support to process overlapping gravitational wave signals. HPC architectures should idealize parallel computing with concepts like cluster computing at their center. GPU Acceleration and Hybrid CPU-GPU Systems are necessary to enhance the performance of intensive tasks like MCMC simulation while maintaining CPU flexibility with GPU efficiency to optimize resource utilization [54, 81].

This is also in the development phase, but these ideas are at the center of the design and implementation of the GlobalFit algorithm.

3.4.6 Specifics in Deploying GlobalFit-like architecture to a DDPC-like environment

The DDPC network comprises various DCCs with heterogeneous hardware configurations. This diversity necessitates the development of adaptable frameworks capable of dynamically allocating resources based on availability and workload demands. The challenge lies in creating a system that can efficiently utilize the varying computational capabilities of different DCCs while maintaining optimal performance across the entire network [31].

A significant hurdle in deploying GlobalFit to a DDPC environment is transitioning from tightly coupled MPI-based systems to more flexible, modular architectures. This shift is crucial for supporting distributed execution without rigid dependencies. The challenge extends to incorporating checkpoint and resume capabilities, essential for handling interruptions in long-running computations. These features are significant given the extended duration of GlobalFit algorithms, which can run for weeks [9].

Despite these challenges, there are examples of successful HPC optimizations in similar astrophysical projects that can provide valuable insights for GlobalFit deployment. The SPACE Center of Excellence has demonstrated significant advancements in redesigning astrophysical codes for exascale systems. Their work has achieved notable performance improvements by implementing innovative algorithms and utilizing heterogeneous computing setups [81].

Another exemplary project is SPOKE 3, which has optimized HPC workflows for astrophysical simulations. This project successfully integrated machine learning and big data analytics techniques to manage and process large datasets effectively. The approaches used in SPOKE

3 could be adapted to address some of the data management challenges faced in the GlobalFit DDPC environment [10].

3.5 Container Technologies in HPC Environments

The integration of container technologies like Docker and Kubernetes into traditional HPC environments has gained significant traction in recent years. This convergence combines containers' flexibility and portability with supercomputers' raw computational power, presenting opportunities and challenges for the HPC community.

3.5.1 Integration of Docker and Kubernetes in Traditional HPC Setups

The need for greater flexibility and reproducibility in scientific computing workflows has driven the adoption of Docker and Kubernetes in HPC environments. Traditional HPC setups typically rely on batch scheduling systems like Slurm or PBS, which are optimized for managing large-scale parallel jobs but lack the agility and portability offered by containerization [11].

To bridge this gap, efforts have been made to integrate container technologies with existing HPC infrastructure. For instance, some HPC centers have implemented Kubernetes clusters that can interface with traditional job schedulers, allowing containerized workloads to be submitted and executed alongside conventional HPC jobs [52]. This hybrid approach enables researchers to leverage the benefits of containers while still utilizing the full capabilities of HPC systems. To better illustrate, Figure 3.4 shows a solution developed by the National Energy Research Scientific Computing Center (NERSC) for HPC services that integrates traditional HPC with container image solutions [53].

However, integrating containers into HPC environments is not without challenges. One significant hurdle is ensuring containerized applications can access specialized hardware and high-performance interconnects commonly found in supercomputers. To address this, custom container runtimes and plugins have been developed to enable containers to efficiently utilize technologies like InfiniBand and GPUs [11].

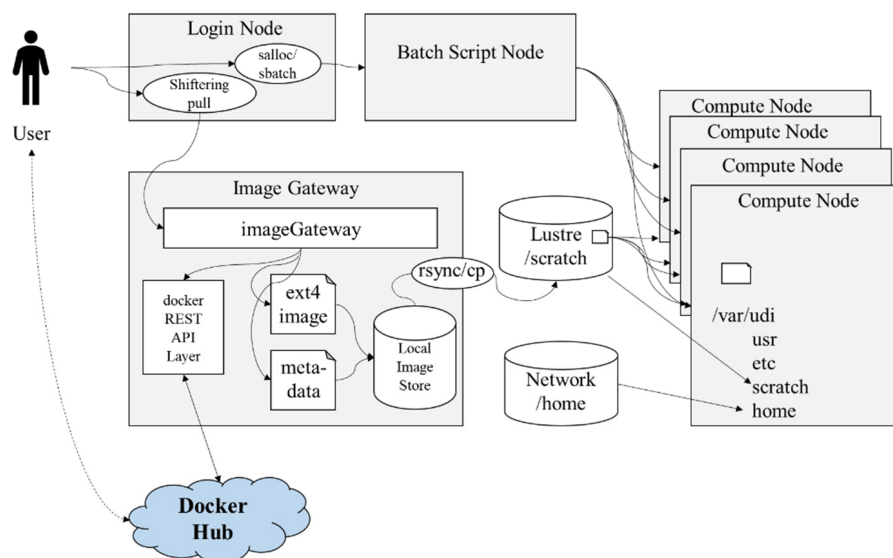


Figure 3.4: NERSC Shifter workflow [53].

3.5.2 Performance Optimization of Containerized Applications on Supercomputers

While containers offer numerous advantages, there have been concerns about their performance overhead in HPC contexts. Recent studies have shown that with proper optimization, containerized applications can achieve near-bare-metal performance on supercomputers [11]. Key strategies for optimizing containerized HPC applications include:

1. Utilizing high-performance container runtimes specifically designed for HPC environments.
2. Enabling direct access to low-level hardware features and high-speed interconnects.
3. Optimizing container images to minimize overhead and maximize performance.
4. Leveraging hardware-specific optimizations and libraries within containers.

Research has demonstrated that when these optimizations are applied, the performance gap between containerized and native HPC applications can be reduced to negligible levels for many workloads [11].

3.5.3 Orchestration Systems for Distributed Scientific Workflows

The complexity of modern scientific workflows has led to the development of specialized orchestration systems that can manage distributed computations across HPC resources. While Kubernetes has emerged as a popular choice for general-purpose container orchestration, HPC-specific solutions have also been developed to address the unique requirements of scientific computing [39]. These HPC-oriented orchestration systems often provide features such as:

1. Integration with existing HPC job schedulers and resource managers.
2. Support for complex workflow dependencies and data management.
3. Efficient handling of large-scale parallel jobs and MPI applications.
4. Mechanisms for checkpointing and restarting long-running computations.

Combining container orchestration's flexibility with HPC-specific optimizations, these systems aim to provide a more seamless experience for researchers working with containerized scientific applications on supercomputing infrastructure [39].

3.5.4 Security Considerations for Containers in High-Performance Environments

When introducing containers into HPC environments, as these systems often deal with sensitive scientific data and valuable computational resources, the shared kernel model of containers presents unique security challenges compared to traditional virtualization approaches used in HPC [84].

HPC centers have implemented multi-layered security strategies for containerized workloads. Strict image validation and signing policies are among those that ensure the integrity of container images. Implementing fine-grained access controls and network segmentation for containerized applications is also standard. Enhanced monitoring and logging capabilities to detect and respond to security incidents, and regular security audits and vulnerability assessments of container infrastructure are crucial for HPC centers [84].

Additionally, some HPC centers have explored the use of specialized container runtimes that provide stronger isolation guarantees, bridging the gap between traditional containers and virtual machines in terms of security [84].

3.5.5 Comparisons Between Traditional Supercomputing Environments and Modern Container-Based Solutions

Adopting container technologies in HPC has sparked debates about the relative merits of traditional supercomputing environments versus modern container-based solutions. While both approaches have their strengths, container-based HPC solutions offer several advantages:

1. Improved reproducibility and portability of scientific workflows.
2. Greater flexibility in software stack management and version control.
3. Easier collaboration and sharing of computational environments among researchers.
4. Potential for more efficient resource utilization through better application isolation.

However, traditional HPC environments still maintain some advantages, particularly in scenarios requiring extreme performance optimization or specialized hardware configurations [74].

The future of HPC likely lies in hybrid approaches that combine the best aspects of both paradigms. By integrating container technologies into traditional HPC environments, researchers can benefit from the flexibility and portability of containers while still leveraging the raw computational power and specialized resources of supercomputers [52, 74].

3.6 Research Gaps and Challenges Identified

The literature review revealed several critical gaps in existing research that justify the need for novel approaches to LISA’s computational challenges. Most current implementations rely on tightly coupled systems using MPI, which proves insufficient for LISA’s distributed computing requirements, where impact isolation between blocks is essential.

The complexity of LISA’s data analysis requirements, involving simultaneous global fitting of multiple astrophysical sources with hundreds of parallel branches over extended periods, exceeds the capabilities of standard computing center queues. This necessitates innovative frameworks capable of satisfying these constraints while minimizing integration efforts and leveraging scientific activities across multiple laboratories.

Furthermore, the heterogeneous nature of the DDPC network, spanning multiple countries with varying HPC architectures and security requirements, demands flexible protocols that adapt to diverse computational environments.

Chapter 4

Globalfit Framework

The GlobalFit framework represents the cornerstone of LISA’s data analysis strategy, providing a comprehensive, modular, and distributed approach to extracting gravitational wave signals from highly complex and overlapping astrophysical sources. Designed to address the LISA mission’s unprecedented computational and organizational challenges, GlobalFit enables simultaneous parameter estimation for thousands of sources and instrument noise models, leveraging advanced MCMC algorithms and high-throughput computing principles. This framework is architected to operate efficiently across a network of international HPC centers, ensuring scalability, flexibility, and resilience in a heterogeneous, collaborative environment.

Due to the partnership with ESA and the large-scale project, the aim is not to find a solution but to contribute to it. This makes it so our contribution is to try to deploy the current prototype version of the whole GlobaFit architecture in a Portuguese HPC center. Since each HPC center is different and has limitations, deploying the framework requires understanding those limitations and making changes in both the architecture and the configuration of the HPC itself. These efforts are so that Portugal is on par with European nations, such as France, the Netherlands, and Germany.

Since this project is still in early stages, we have multiple prototypes of the Globalfit framework from various countries, e.g., the USA, France, and Germany. This thesis focuses on the French version because it offers a modular architecture and robust checkpoint/resume functionality, which aligns closely with our requirements for flexible integration and long-duration computation in distributed HPC environments.

The methodology adopted for deploying and validating the GlobalFit framework followed a phased approach. In this chapter, we will dive into the first step, which is understanding the framework and local deployment on a regular computer. The following chapter will outline the subsequent phases, which briefly consist of: deploying the framework isolated in an HPC, and only after trying to deploy the whole DDPC architecture in the same chosen HPC.

4.1 Globalfit Framework

Because real LISA data is not yet available at this stage, the GlobalFit framework is being prototyped using a simple simulated input—a Gaussian curve. This approach allows us to test and validate the framework’s core functionality by performing fits on this artificial data, making the computation trivial and ensuring that the focus remains on the development and evaluation of the workflow infrastructure rather than on complex scientific analysis. The main objective is to establish and demonstrate the advanced process control (APC) and high-throughput computing (HTC) capabilities of the current working version in a real HPC environment, rather than to achieve meaningful scientific results at this point.

The APC implementation of the GlobalFit framework is a high-level workflow manager for Python-based jobs, tailored for HTC. Its primary goal is to automate and scale up workflows that would otherwise require manual intervention, enabling the execution of thousands of tasks (blocks) across multiple iterations on local or cluster resources.

4.1.1 Core Concepts of the APC

The framework is at version 0.4.0 and can fulfill the following requirements.

1. **Blocks:** The fundamental operational units represent a computational task expected to run for minutes or hours. Each block can coordinate with other blocks, wait for dependencies, spawn new tasks, and manage its inputs/outputs.
2. **Iterations:** Cycles over which blocks are repeatedly executed, typically lasting minutes to hours. The infrastructure is optimized for slow, coarse-grained tasks, not high-frequency or fine-grained operations like those inside MCMC samplers.
3. **Communication between blocks:** Blocks communicate via an event bus; in this case, we use a NATS messaging bus. This allows for fast inter-block communication without having to handle concurrency problems.
4. **Python-Centric:** The workflow manager natively executes Python code. For other languages (C, C++, Julia), integration is achieved via Python wrappers or by invoking external executables from within blocks.

Since this tool must operate within the DDPC network, some classes communicate externally with components used to deploy the whole infrastructure on an HPC center. We will discuss these components and highlight them in the following section when we deploy the framework.

All these features are possible due to a complex abstraction from any defined structure using Python base classes to allow any configuration input by the user to work. The Figure 4.1 highlights the complexity of this tool, showing a type of class diagram with the interactions of different components of this tool.

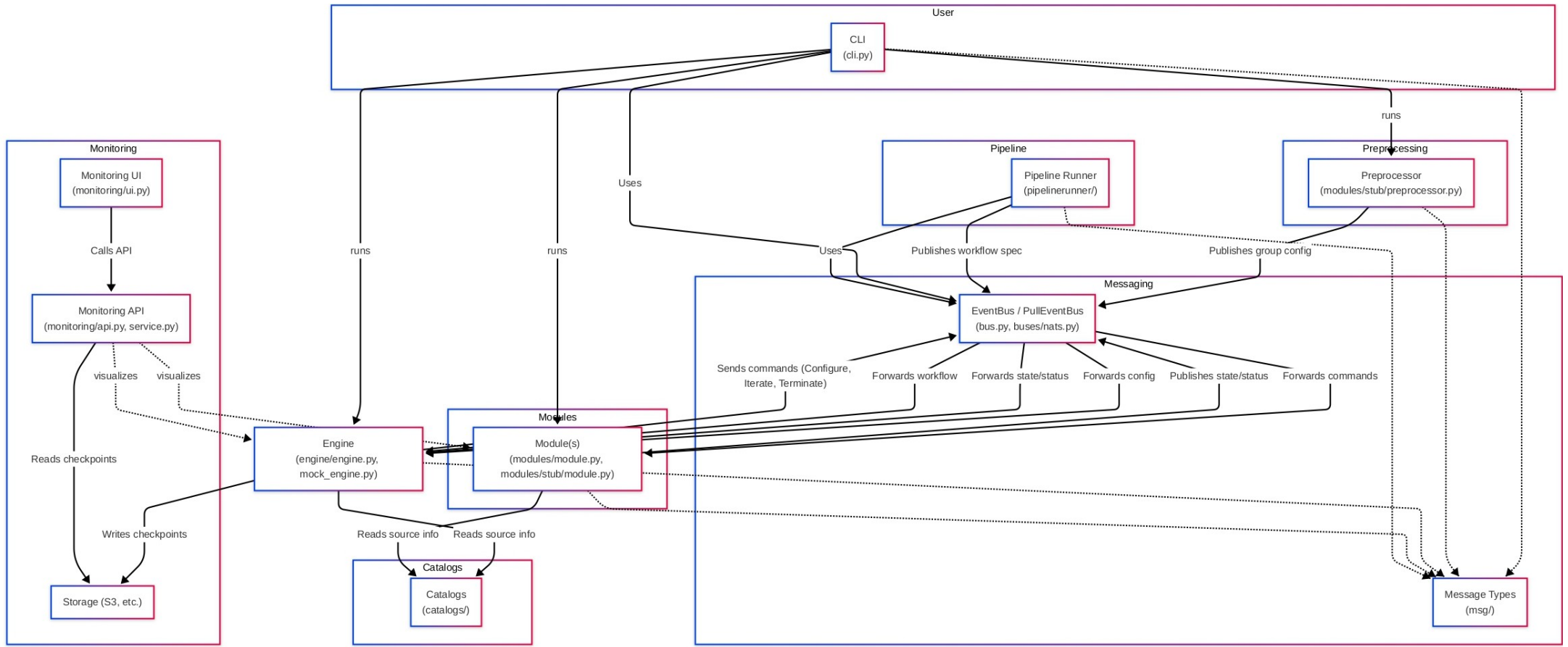


Figure 4.1: Globalfit APC class diagram.

To better illustrate the complex interactions within the GlobalFit framework, we will analyze Figure 4.1 and break it into two complementary diagrams highlighting different aspects of the architecture, the communication Flow, and the Component Roles and Responsibilities. This will, in turn, make it simpler to understand Figure 4.1.

4.1.2 Communication Architecture

The GlobalFit framework employs a sophisticated event-driven architecture centered around an event bus that facilitates communication between all components. This architecture ensures loose coupling between components while maintaining efficient message passing capabilities.

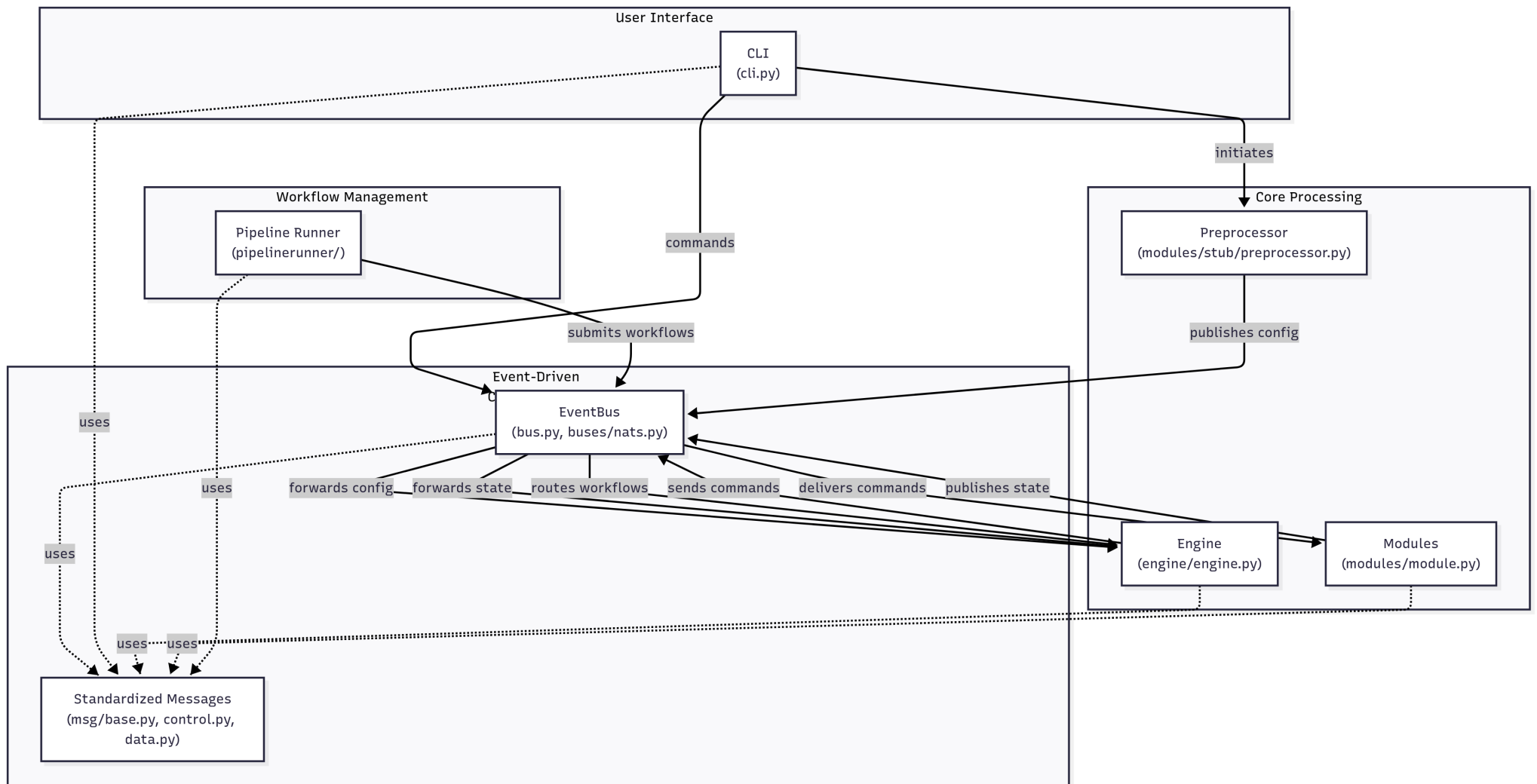


Figure 4.2: Communication Architecture diagram.

Figure 4.2 illustrates the communication flow within the GlobalFit framework, emphasizing the central role of the event bus in mediating all inter-component communication. The diagram shows how commands, configurations, and results flow through the system using standardized message types, ensuring reliable and consistent communication patterns. The following subsections will discuss and explain Figure 4.2.

4.1.2.1 Event Bus Communication Pattern

The core communication mechanism relies on implementing EventBus, which provides publish/-subscribe messaging patterns using NATS as the backend. All inter-component communication flows through this central messaging system, including module-to-engine communication where modules publish their state, results, and execution status updates to the engine via the event bus.

The engine-to-module communication involves the engine sending configuration commands, iteration requests, and termination signals to modules. Preprocessor integration occurs when the preprocessor publishes module group configurations consumed by the engine. Workflow orchestration is handled through the pipeline runner, submitting workflow specifications and handling results through the event bus.

4.1.2.2 Message Types and Subjects

The framework defines standardized message types in the **msg/** module, including control messages such as ConfigureModule, IterateModule, and TerminateModule commands. Data messages encompass ModuleState, ExecutionStateUpdate, and result publications. Configuration messages include ModuleGroupConfiguration for module setup.

This standardized messaging ensures consistent communication protocols across all components and enables reliable message routing and processing.

4.1.3 Component Roles and Responsibilities

The GlobalFit framework architecture comprises ten primary components with specific roles and responsibilities in the computational workflow.

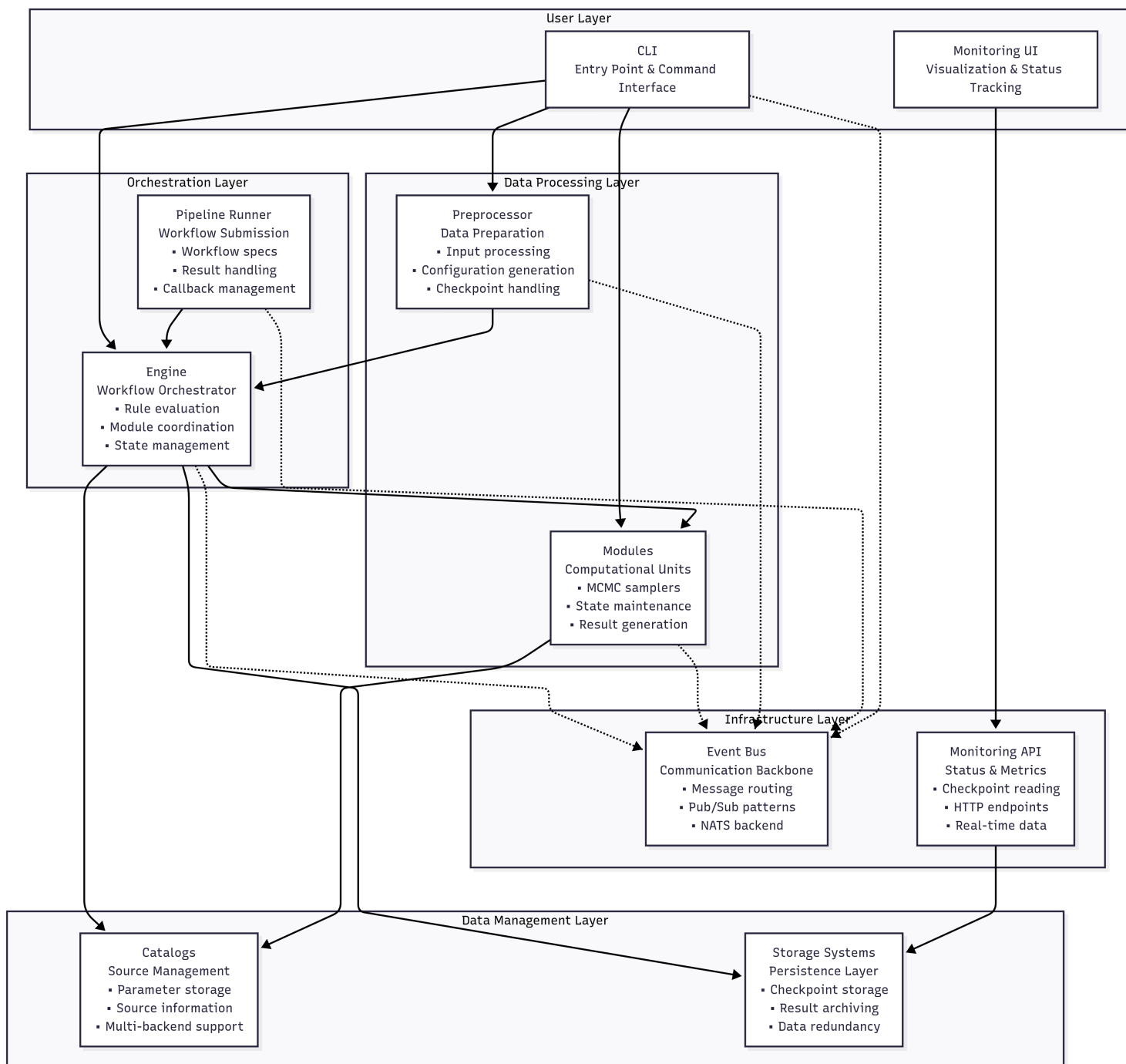


Figure 4.3: Component Roles and Responsibilities diagram.

Figure 4.3 presents the component roles and responsibilities, organized into logical layers that demonstrate the hierarchical structure of the framework. Each layer serves specific functions: the user layer provides interfaces, the orchestration layer manages workflows, the data processing layer handles computations, the data management layer ensures persistence, and the infrastructure layer provides the underlying communication and monitoring capabilities. In the next sub-subsections, we will discuss and explain Figure 4.3.

4.1.3.1 Core Computational Components

Modules serve as the fundamental computational units, representing individual MCMC samplers or other analysis algorithms. Each module maintains its state, processes data independently, and communicates results through the event bus. The modular design allows different laboratories to implement specialized algorithms using various programming languages while maintaining compatibility with the overall framework.

The engine acts as the central orchestrator, managing the execution of modules and coordinating the overall workflow. It handles rule evaluation, collects results from modules, manages module states, and makes decisions about iteration progression. The engine subscribes to module status updates and sends commands to modules based on the current workflow state.

4.1.3.2 Data Processing and Management

Preprocessors are responsible for preparing input data and generating initial module configurations. They can utilize checkpoints to resume processing or adjust configurations based on previous results. The preprocessor writes configuration files and publishes group configuration messages to initiate module execution.

Catalogs manage source catalogs using various backends (memory, SQLite) and provide source parameters to modules and the engine. This component ensures efficient access to the source information required for analysis.

4.1.3.3 Workflow and Monitoring

Pipeline Runner handles workflow submission and orchestration, bridging workflow definitions and the event bus. It publishes workflow specifications and listens for workflow results and callbacks, enabling complex multi-step analysis pipelines.

Monitoring provides comprehensive APIs and user interfaces for tracking the state and results of workflows and modules. It reads checkpoints and status information from storage systems (e.g., S3) and exposes HTTP endpoints for real-time monitoring and visualization.

4.1.3.4 Supporting Infrastructure

CLI serves as the primary entry point for users, parsing arguments and invoking appropriate commands. It provides interfaces for running workflows, managing modules, and controlling preprocessing operations.

Event Bus abstracts the message passing infrastructure, implementing publish/subscribe patterns using NATS. This component ensures reliable message delivery and provides the foundation for all inter-component communication.

Messaging System defines the standardized message types and communication protocols used throughout the framework, ensuring consistent data exchange between components.

4.1.4 Engine and Modules

After breaking down Figure 4.1 in the two Figures 4.2 and 4.3 and explaining each in its context, we can interconnect them and understand how the engine and modules interact. If we isolate the connection between the engine and the multiple modules in all three pictures, we achieve a result like—figure 4.4.

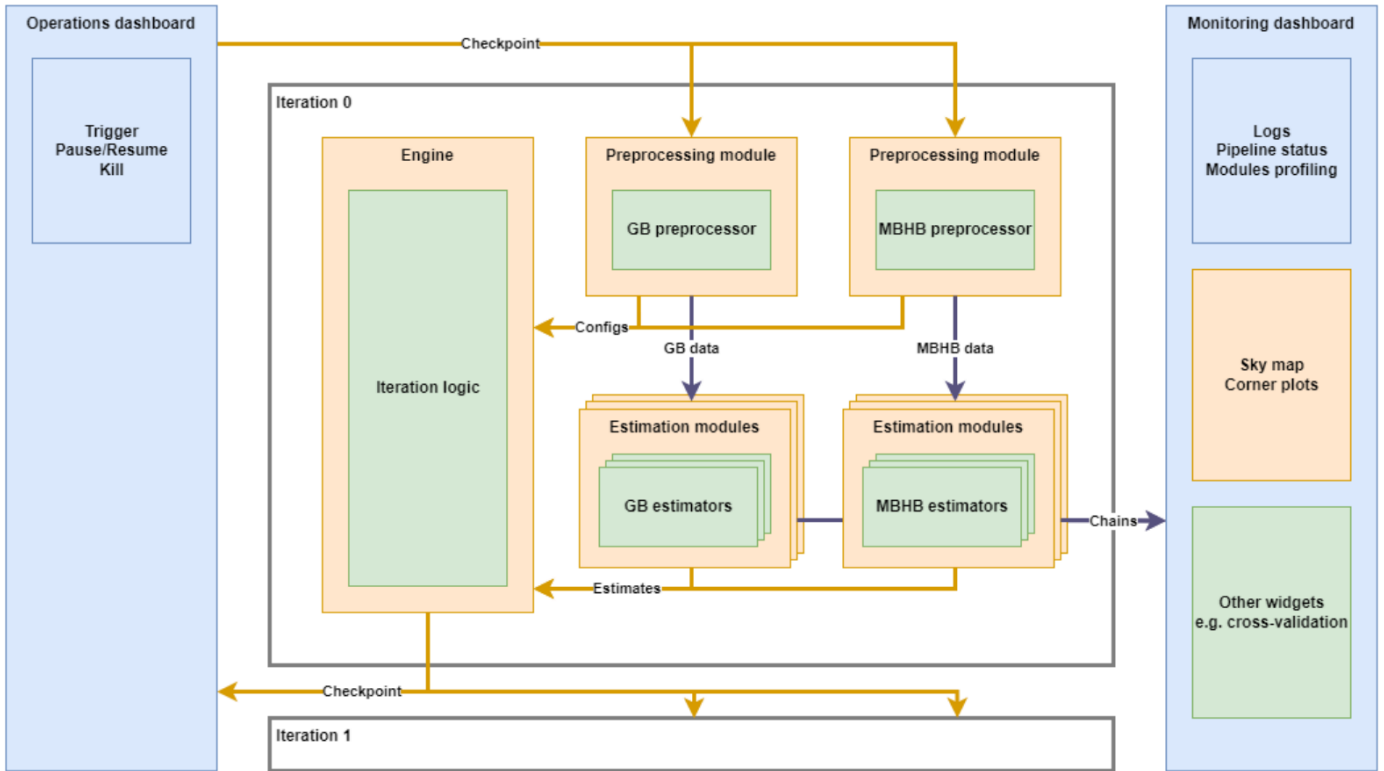


Figure 4.4: Engine and Modules connection in the GlobalFit

Figure 4.4 illustrates the interaction between the Engine and Modules within the GlobalFit framework, highlighting the modular and orchestrated nature of the system. The engine is the central orchestrator, responsible for managing the execution flow, sending commands, and collecting results from each module. Each module encapsulates a specific computational task—such as a specialized estimator—and operates independently, like the MBHB preprocessor, where we can see an example result in Figure 2.1, processing data and maintaining its state. As we saw before, an event-driven messaging system facilitates communication between the Engine and Modules, ensuring that configuration commands, iteration requests, and results are exchanged efficiently and reliably.

This architecture enables loose coupling between components, allowing different laboratories to develop and integrate their Modules in various programming languages, as long as they adhere to the standardized messaging protocols. The engine coordinates the workflow, dynamically managing the Modules' lifecycle, handling dependencies, and ensuring synchronization across

iterations. This design supports scalability and flexibility, enabling the framework to adapt to scientific requirements and computational environments.

4.2 Local Deployment

Based on the figure 4.1 and the explorations described in the previous section, we could create and share documentation A, which was previously unavailable within the project. This enabled us to successfully deploy the GlobalFit framework locally, as detailed in Annex D.

4.3 Challenges

One major technical challenge was reconciling the diverse technologies integrated within the GlobalFit framework. The system includes a Python-based workflow manager (APC), an event-driven messaging layer (NATS), and various computational modules that may be implemented in different languages. Understanding how these components interact required a comprehensive study.

Given the nascent state of the GlobalFit framework documentation, a substantial portion of the research effort was dedicated to creating comprehensive documentation for future researchers and developers. This process required understanding the technical implementation details and articulating the conceptual frameworks and design decisions that informed the system architecture. Several classes and modules were sparsely documented, necessitating careful reading and interpretation of the source code to infer intended behaviors and interfaces. This reverse-engineering process was critical for clarifying how dependencies are managed and inputs and outputs are passed between modules. Constructing accurate architectural diagrams (such as Figures 4.1, 4.2, 4.3) was closely tied to this effort; each diagram element was validated against the implementation. Systematic verification of the implemented logic ensured that the diagrams faithfully represented the actual architecture.

Fully comprehending the GlobalFit system's architecture and implementation details required a substantial upfront investment of time. Tasks include studying design notes, examining multiple prototype codebases, and experimenting with local deployment. We carefully planned tasks in iterative phases, allocating sufficient time for exploration and testing. This experience underscored that complex systems often demand significant initial exploration and learning.

4.3.1 Lessons learned

In response to these difficulties, a thorough and methodical approach was adopted. We undertook an extended analysis of system logs and configurations and incremental testing of each integration step. This meticulous process helped uncover subtle issues and ensure each component interacted as intended. Rigorous cross-checking and validation processes maintained alignment between framework implementation and documentation, establishing confidence in system correctness and architectural consistency. Given this, we can take away the following lessons:

- **Thorough analysis:** Careful examination and testing of each component interface are essential when integrating multiple, diverse technologies into a single system.
- **Systematic documentation:** Consistent, up-to-date documentation (including clear diagrams) helps maintain clarity and fidelity between design and implementation.
- **Incremental integration:** Careful planning of integration steps and incremental validation of results improve manageability and robustness when combining complex software components.

Given that we successfully validated and deployed the GlobalFit framework, we are now ready to dive into the next chapter, where we will attempt to run this framework in a real HPC environment, in this case, the Decaulion supercomputer.

Chapter 5

Deucalion Supercomputer

This chapter aims to present the HPC platform Deucalion in the context of its integration within the DDPC network and assess its suitability for hosting and executing the GlobalFit framework. Given the framework’s increasing complexity and the scale required by the LISA mission, selecting an adequate HPC environment is of particular importance. Deucalion has emerged as a strategic partner for HPC to explore deployment and integrate Portugal into the DDPC network. All work developed for the project was under the project code name HPCvLAB@UPORTO from the University of Porto, and we were the first ones from the University of Porto to have access to and use Deucalion’s resources.

This chapter begins with an overview of Deucalion’s positioning within the EuroHPC initiative, highlighting its architectural innovations and computational capabilities. This is followed by a detailed description of the system’s architecture, hardware specifications, and software environment. Special attention is given to Deucalion’s containerization capabilities—most notably its support for Singularity—as they are crucial for deploying the modular and portable workloads required by the GlobalFit framework.

The motivation for exploring containerization and toolchain support on Deucalion is closely tied to the framework’s implementation requirements. The GlobalFit architecture heavily relies on encapsulating software dependencies and ensuring consistent execution across heterogeneous environments. Thus, the ability to run containerized applications efficiently and securely becomes essential. This is particularly relevant in a DDPC setting, where compute nodes across different countries and vendors must run identical workloads under varying local constraints.

Furthermore, since GlobalFit is designed to be modular and adaptable across laboratories and scientific institutions, tools such as Singularity, Slurm, and environment module systems must be evaluated in terms of their integration capabilities. Therefore, this chapter not only documents Deucalion’s technical profile but also lays the groundwork for understanding the practical feasibility of deploying the GlobalFit pipeline in a real-world HPC environment.

5.1 EuroHPC Initiative and Deucalion's Role



Figure 5.1: Deucalion supercomputer.

The European High Performance Computing (EuroHPC) Joint Undertaking has co-funded and deployed a network of leading supercomputers across the EU to prepare for exascale computing. Deucalion is the seventh EuroHPC system (after Discoverer, MeluXina, Vega, Karolina, Leonardo, and LUMI) [21]. Located at the Minho Advanced Computing Centre (MACC) in Portugal, Deucalion was inaugurated in 2023 with a peak performance of about 10 petaflops (10 million billion FLOPS) [21]. Its architecture is unique among EuroHPC machines: it combines Fujitsu PRIMEHPC A64FX (ARM) CPUs and Atos Bull Sequana (AMD EPYC) nodes [21], making it the first EuroHPC cluster to employ ARM processors. This heterogeneity enriches Europe's computing architecture portfolio and aligns with the European Processor Initiative (EPI) goals of energy-efficient CPUs [21]. Deucalion's heterogeneous architecture includes a GPU-accelerated partition of 33 nodes equipped with NVIDIA GPUs for artificial intelligence and machine learning workloads [3, 21]. Deucalion is co-funded by EuroHPC and Portugal (FCT) and is rated among the first generation of EuroHPC pre-exascale petascale supercomputers[3, 21].

Deucalion is intended for "green" high-performance research across domains (climate, AI, materials, astrophysics, etc.) [21]. Its 10 petaflops capacity and heterogeneous design mean it can serve various scientific workloads. For example, Deucalion's mix of ARM and x86 processors complements the existing EuroHPC systems. It offers a unique testbed for porting and optimizing software (including from the European microprocessor development effort) [3, 21]. The new

architecture thus complements standard CPU clusters and even GPU-based systems, broadening Europe’s HPC ecosystem [21].

5.2 System Architecture and Hardware Specifications

Deucalion features a unique heterogeneous architecture comprising three distinct computing partitions, each optimized for different computational workloads [30, 92]. This heterogeneous design distinguishes Deucalion as the first European supercomputer to incorporate ARM and x86 micro-processors alongside GPU acceleration within the same cluster [57].

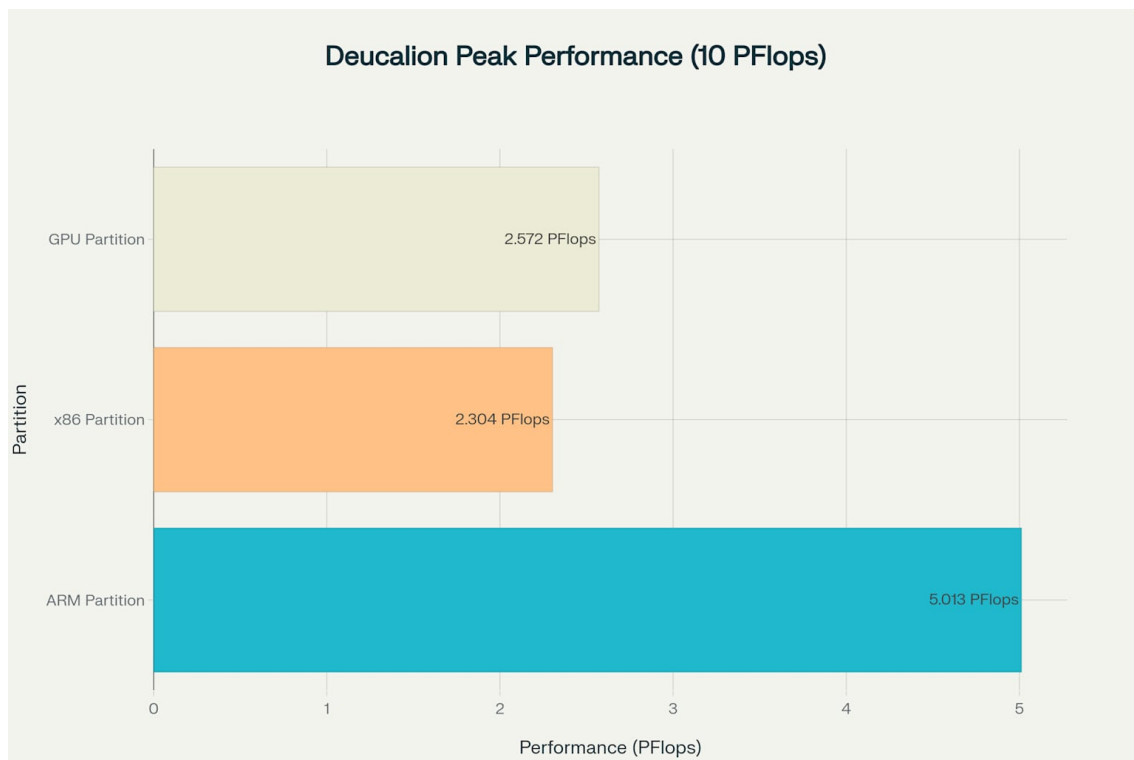


Figure 5.2: Distribution of peak computational performance across Deucalion’s three main partitions, showing the heterogeneous architecture of Portugal’s flagship supercomputer [57].

The ARM partition constitutes the most significant component of Deucalion, consisting of 1632 Fujitsu PRIMEHPC FX700 nodes equipped with Fujitsu A64FX 2.0 GHz processors [30, 32, 92]. Each ARM node provides 48 cores and 32GB of High Bandwidth Memory, contributing a peak performance of 5.013 PetaFLOPS and totaling 78,336 cores with 52TB of combined memory [32, 57]. This ARM-based partition leverages the same processor architecture used in Fugaku, previously the world’s fastest supercomputer, demonstrating energy efficiency and performance characteristics well-suited for scientific computing applications [29, 30].

The x86 partition comprises 500 ATOS Bull Sequana X440 dual-processor nodes, each equipped with two AMD EPYC Rome 7742 processors running at 2.25 GHz [30, 32]. With 128 cores per node and 256GB of DDR4 memory, this partition delivers 2.304 PetaFLOPS peak performance

across 64,000 cores and 128TB of total memory [30, 57]. The x86 architecture ensures compatibility with a broad ecosystem of scientific software and applications that require traditional processor architectures [57].

The GPU-accelerated partition consists of 33 ATOS Bull Sequana E410 nodes, divided into two configurations: 17 nodes with NVIDIA A100 40GB GPUs and 16 nodes with NVIDIA A100 80GB GPUs [30, 32]. Each GPU node contains two AMD EPYC Rome 7742 processors and four NVIDIA A100 GPUs, providing 512GB of system memory and delivering a combined GPU performance of 2.572 PetaFLOPS [30, 32]. This GPU partition is specifically designed for artificial intelligence, machine learning, and computationally intensive scientific applications requiring massive parallel processing capabilities [3].

The following diagram (from the official documentation) illustrates the overall Deucalion architecture and network:

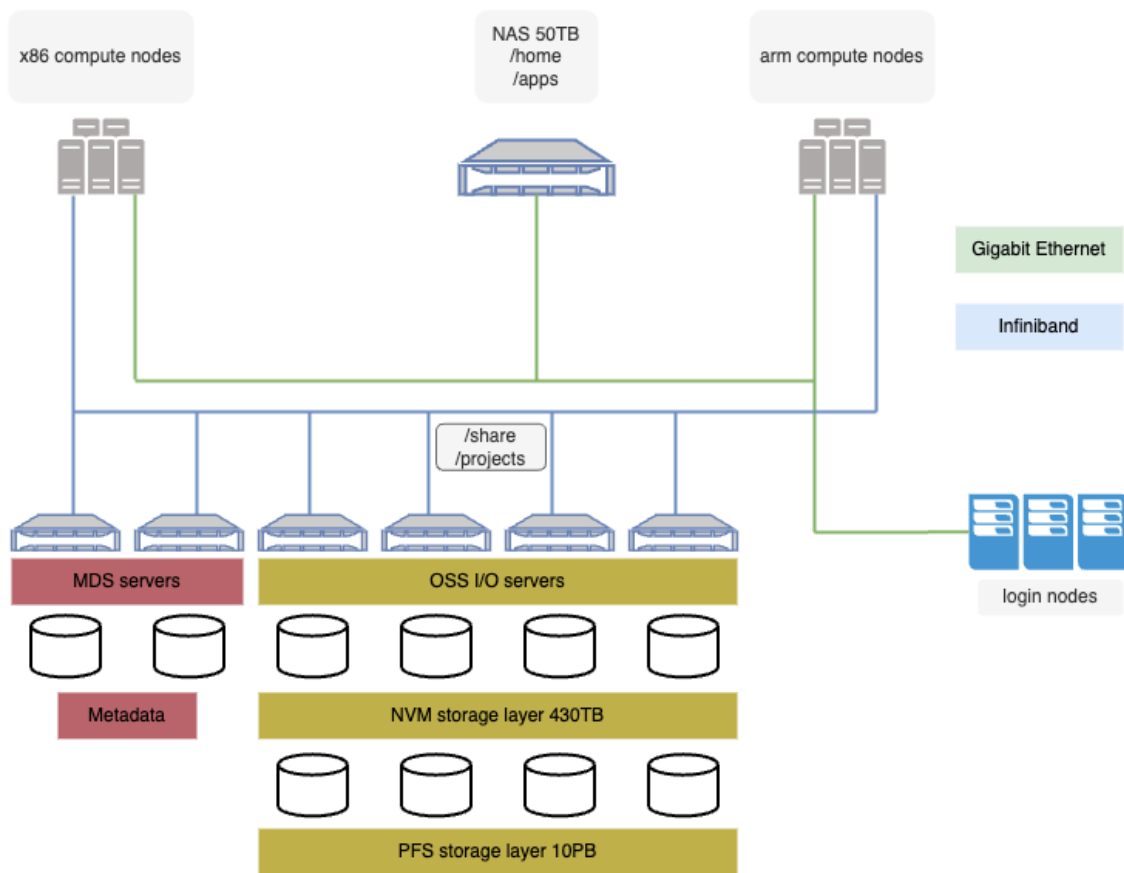


Figure 5.3: Simplified architecture of the Deucalion supercomputer. ARM and x86/GPU compute nodes (left/right) connect via high-speed InfiniBand to a Lustre parallel file system (center). Login and I/O traffic use Ethernet and NFS. [57].

Blue arrows denote InfiniBand (IB) connections in Figure 5.3. Each ARM and GPU/X86 node has an NVIDIA ConnectX-6 network interface (100Gb/s for ARM nodes, 200Gb/s for x86/GPU nodes) [57]. The ARM partition's IB network is a fat-tree with 1:1.6 oversubscription, using

50 Mellanox QM8790 switches; the x86/GPU partition has a non-blocking fat-tree (1:1) with 27 switches [57]. Practically, any x86 or GPU node can talk to any other at full 200Gb/s bandwidth, while ARM nodes share connectivity (approximately 100Gb/s) in a blocked network. This hybrid IB fabric ensures low latency and high bandwidth for MPI and RDMA traffic across all nodes. In addition to InfiniBand, Deucalion has an Ethernet network (Mellanox AS4610 top-of-rack and SN2410/SN2010 core switches) for cluster management and to serve a 50TB SSD-backed NAS via NFS for the */home* directory. For the */projects*, it's a DDN Lustre system [57].

5.2.1 Partition Selection and Rationale

For the deployment of the GlobalFit framework on Deucalion, we specifically utilize the ARM partition. This choice is informed by the architecture of the framework and the requirements of our computational workloads. The ARM partition is Deucalion's largest and most significant component, comprising 1,632 Fujitsu PRIMEHPC FX700 nodes with A64FX processors, totaling over 78,000 cores and 52TB of memory. This partition leverages the same processor technology as Fugaku, formerly the world's fastest supercomputer, and is optimized for high-throughput scientific computing tasks.

Since the GlobalFit framework is written in Python and relies on the parallel execution of many independent tasks, it is well-suited for the ARM nodes' high core count and memory bandwidth. And while Deucalion also offers x86 and GPU partitions, our current scientific objectives do not require GPU acceleration. The x86 partition is smaller and primarily intended for workloads with different compatibility requirements. We focus on the arm partition.

5.3 Software Environment and Resource Management

Deucalion runs a standard HPC software stack. All compute nodes boot into Rocky Linux 8 (a RHEL-compatible OS) [57]. The administrators provide Environment Modules (Lmod) for managing the software environment. In addition, the cluster supports EESSI (the European Environment for Scientific Software Installations) to provide a federated library collection: EESSI modules are available on ARM and GPU partitions (with about 30% coverage on ARM, more are planned to come in the future) [57]. Researchers can load standard packages (MPI, BLAS, Python, etc.) via EESSI or the native module repository. This flexibility helps port codes seamlessly between architectures (ARM vs x86) and other EuroHPC sites.

The job scheduler is Slurm (version 23.11.4) [57]. Users submit batch jobs or interactive allocations via Slurm queues. The cluster also offers Open OnDemand (OOD) for web-based access and job monitoring. For example, typical Slurm submissions specify the partition (normal-arm, normal-x86, or gpu), number of tasks, and memory. Deucalion's documentation includes examples of running MPI or Python programs on each partition [57]. Being a traditional HPC environment, all nodes use a shared-user identity and network filesystem; there is no support for container orchestration systems like Kubernetes.

5.3.1 Containerization

Deucalion supports containerized applications only via Singularity (now known as Apptainer). The cluster provides Singularity 3.11 on all partitions [57]. Docker itself is unavailable on compute nodes (since Docker requires root privileges and is generally disallowed on HPC). However, Singularity is designed for HPC: it can import and run Docker/OCI images without requiring admin privileges [57]. In practice, users build or pull SIF and run them under Slurm (e.g., `srun singularity exec ...`). The documentation emphasizes that Singularity offers portable, reproducible environments and native MPI support (including IB drivers) [57].

One constraint is multi-architecture: a container must be built on or for the same CPU architecture as the target partition. For instance, a Docker container built on x86 cannot run on the ARM nodes without rebuilding. The user guide even provides Singularity definition files (`openmpi_arm.def` vs `openmpi_x86.def`) to create custom images with appropriate MPI libraries and Mellanox drivers [57]. In summary, containerized workflows are supported, but the workflow developer must provide Singularity images and possibly rebuild images per architecture. The lack of native Docker/OCI runtimes means that container portability relies entirely on Singularity's conversion or on distributing SIF images. This means there is no Kubernetes support, nor is it installed on the system, since Kubernetes works only with OCI container engines like Docker or Podman, as seen previously.

5.4 Globalfit framework in Deucalion

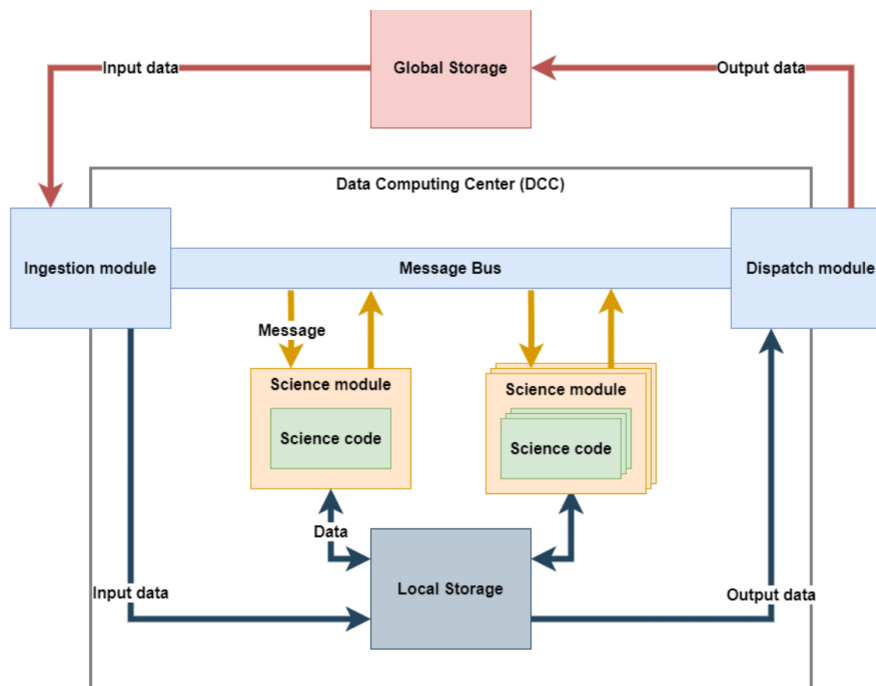


Figure 5.4: DCC Architecture [9]

Implementing the GlobalFit framework on Deucalion represents a critical step toward establishing Portugal’s contribution to the LISA DDPC network.

This section details the methodology employed, technical challenges encountered, and solutions developed during deployment, only after we deploy the architecture highlighted in Figure 3.2.

5.4.1 Implementation Strategy and Methodology

The deployment strategy for the GlobalFit framework on Deucalion was designed with a phased approach to ensure systematic validation and risk mitigation. The methodology consisted of two primary phases:

- **Isolated Framework Deployment:** Initial framework deployment in standalone mode to validate basic functionality and compatibility with Deucalion’s heterogeneous architecture.
- **Integrated DCC Architecture Deployment:** Subsequent integration of the complete DCC architecture as depicted in Figure 5.4, enabling full participation in the DDPC network. We will discuss this in the last chapter, as it is the main objective of this thesis.

This phased approach was adopted to address the inherent complexity of integrating a distributed computing framework with a heterogeneous HPC environment while minimizing potential disruptions to ongoing research activities.

5.4.2 Installation and Dependency Management

The installation process revealed several technical challenges that required innovative solutions. All the in-between steps are detailed in-depth and described in the Annex E. Here we will highlight the main steps, results, and conclusions.

Environment Setup Challenges: The standard installation approach using UV from Astral was not feasible within Deucalion’s environment. Consequently, a custom installation methodology was developed using native Python virtual environments. This approach required:

1. Utilization of Slurm’s interactive job allocation for environment setup
2. Installation of dependencies within compute nodes rather than login nodes to ensure architecture compatibility
3. Custom compilation and configuration of Python packages for ARM architecture

Binary Dependencies: The framework’s dependency on the NATS messaging system required manual acquisition and deployment of ARM-compatible binaries. The selection of NATS server version 2.11.3 for Linux-arm64 was based on compatibility requirements and performance considerations.

5.4.3 Challenges and Solutions

The deployment on Deucalion required careful consideration of several architectural factors unique to this heterogeneous supercomputing environment:

Multi-Architecture Compatibility: Deucalion’s heterogeneous design, featuring ARM (Fujitsu A64FX), x86 (AMD EPYC), and GPU partitions, necessitated architecture-specific adaptations. The GlobalFit framework, being Python-based with dependencies on various computational libraries, required careful attention to ensure compatibility across different processor architectures. The solution involved developing architecture-specific installation procedures, maintaining separate binary distributions for other processor architectures, and respecting Deucalion’s temporary file system policies.

Resource Management Integration: Integration with Slurm, Deucalion’s job scheduling system, was essential for proper resource allocation and job management. The framework’s event-driven architecture, utilizing NATS messaging, must operate within Slurm’s job constraints and resource allocation policies. This made it necessary for us to develop efforts to create Slurm-compatible job scripts that properly manage resource allocation.

Containerization Limitations: Deucalion’s support for containerization through Singularity, rather than Docker, required adaptation of the framework’s deployment strategy. This limitation influenced decisions regarding dependency management and runtime environment configuration.

5.4.4 Performance Validation

The successful deployment was validated, as we can see in Annex E through comprehensive performance testing that demonstrated:

Resource Utilization: Testing confirmed that the framework effectively utilized all available CPU cores, achieving 100% CPU utilization during compute-intensive phases, as evidenced by system monitoring data.

Scalability: The framework demonstrated successful scaling from simple test cases (single Gaussian fitting) to more complex scenarios involving multiple sources (2000 Gaussians, 3650 samples), validating its suitability for LISA-scale data processing requirements.

Checkpoint and Resume Functionality: Critical to long-running HPC jobs, the framework’s checkpoint and resume capabilities were successfully validated, ensuring robustness in the face of system interruptions or job scheduling constraints.

5.5 Technical conclusion

The successful deployment of the GlobalFit framework on Deucalion marks a significant milestone for Portugal’s participation in the LISA DDPC network. This accomplishment demonstrates both the adaptability of the GlobalFit framework to diverse HPC environments and Deucalion’s capability to support sophisticated scientific workflows.

Technically, the framework was effectively adapted to Deucalion's heterogeneous architecture, including ARM and x86 partitions. This required innovative solutions to address architecture-specific challenges while maintaining the framework's core functionality. Integrating Deucalion's Slurm-based job scheduling system allowed efficient resource utilization and seamless operation within the existing HPC workflow. The deployment process also involved developing architecture-specific installation procedures and maintaining separate binary distributions to ensure compatibility across different processor types.

After successfully validating and deploying, we are ready to delve into the next chapter, where we will attempt to deploy the current DDPC architecture in Deucalion.

Chapter 6

Deploying the DCC on Deucalion

Having successfully deployed and validated the GlobalFit framework in isolation on Deucalion, this research’s final and most critical phase involved deploying the complete DCC architecture to enable Portugal’s full participation in the LISA DDPC network. This chapter documents the comprehensive analysis of requirements, infrastructure constraints, attempted solutions, and ultimately the fundamental challenges that prevented successful deployment, along with proposed recommendations for future work.

6.1 Current DDPC Architecture Requirements

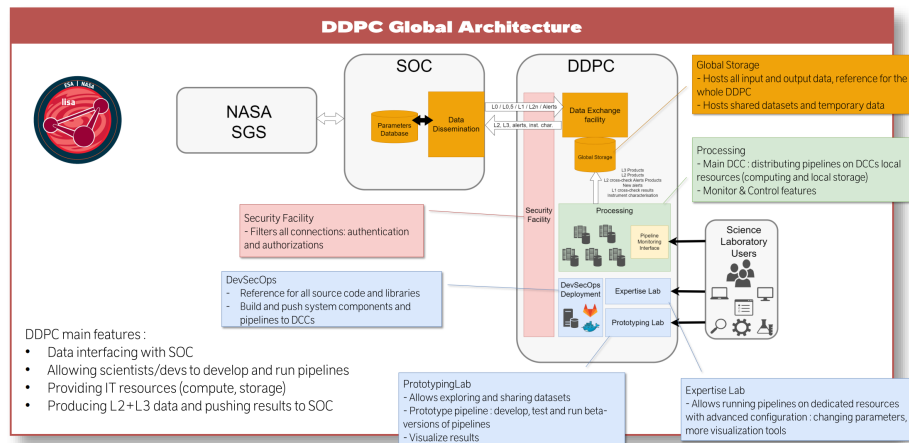


Figure 6.1: DDPC Global Architecture [63]

The LISA DDPC network in Figure 6.1 operates on a sophisticated distributed architecture that bridges multiple High-Performance Computing centers across different countries and institutions. The core architecture centers around a Kubernetes-based orchestration layer that manages containerized workloads and facilitates communication between distributed computing resources.

This architecture has been carefully designed to maintain consistency across heterogeneous computing environments while providing the flexibility needed for the complex computational demands of gravitational wave data analysis. We can visualize this architecture in Figure 6.2.

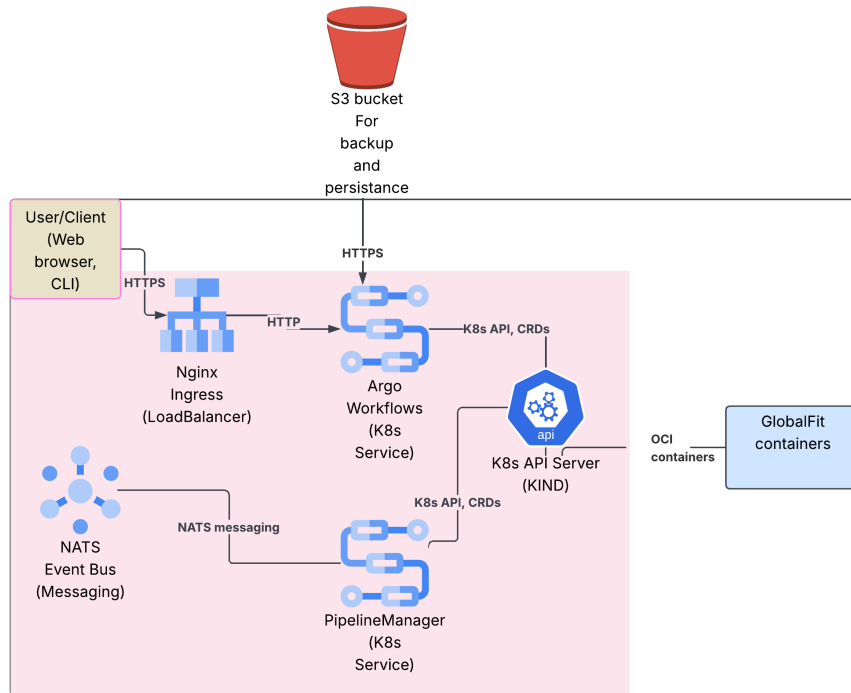


Figure 6.2: DCC Cloud Architecture

The fundamental components of the DDPC architecture require a comprehensive technology stack that includes several interconnected systems. The foundation relies on a Kubernetes cluster deployed using kind (Kubernetes in Docker), which creates local Kubernetes clusters using Docker container nodes. This approach enables the deployment of Argo Workflows as the primary workflow engine for orchestrating complex scientific pipelines and the NATS messaging system for event-driven communication between components. Additionally, the architecture incorporates Nginx Ingress for load balancing and service exposure, Cert-Manager for TLS certificate management, and optional components such as Jupyter notebooks and GlobalFit monitoring interfaces.

6.2 Deucalion Infrastructure Analysis

The technical analysis of Deucalion’s infrastructure revealed strengths and critical limitations that directly impact the deployment of the DDPC architecture. The system operates under Rocky Linux 8 with Slurm as the primary job scheduler and provides comprehensive support for scientific computing workloads through Environment Modules and EESSI software provisioning. However, Deucalion’s containerization capabilities present the primary constraint for DDPC deployment.

The system exclusively supports Singularity as its container runtime, which is specifically designed for HPC environments and provides excellent security and performance characteristics for scientific workloads. While Singularity excels in its intended domain, it fundamentally differs from the OCI-compatible container engines required by Kubernetes orchestration systems. The absence of Docker or Podman container engines on Deucalion creates a barrier for deploying kind-based Kubernetes clusters, which form the foundation of the DDPC architecture.

The infrastructure analysis also revealed that Deucalion's security model and administrative policies prioritize stability and user isolation, which typically prevent the installation of additional container engines that might introduce security vulnerabilities or system instability.

6.3 Prerequisites, Documentation, and Deployment Tool Analysis

Based on the available proprietary documentation, the German team successfully deployed an instance of the LISA DDPC in the Max Planck Computing and Data Facility on the Albert Einstein Institute (AEI) on an HPC that also only had Slurm and Singularity, just like Deucalion. The comprehensive analysis of deployment requirements, based primarily on the German AEI bridge implementation, revealed the extensive prerequisites necessary for successful DDPC deployment. The German team's solution represents the most mature adaptation of the DDPC architecture to HPC environments with Slurm scheduling, making it the most relevant reference implementation for Deucalion deployment. Their approach maintains the core DDPC functionality while adapting to the constraints of traditional HPC infrastructure through the innovative use of a new tool they created called InterLink bridging technology, as shown in Figure 6.3. This technology plays the role of translating Kubernetes pod specifications and workflow tasks into Slurm job submissions, allowing scientific workflows orchestrated in a Kubernetes environment to be executed on HPC resources that do not natively support Kubernetes or standard container engines like Docker or Podman. The interlink uses Singularity for cases where a specific container may need to be run to help the Slurm jobs.

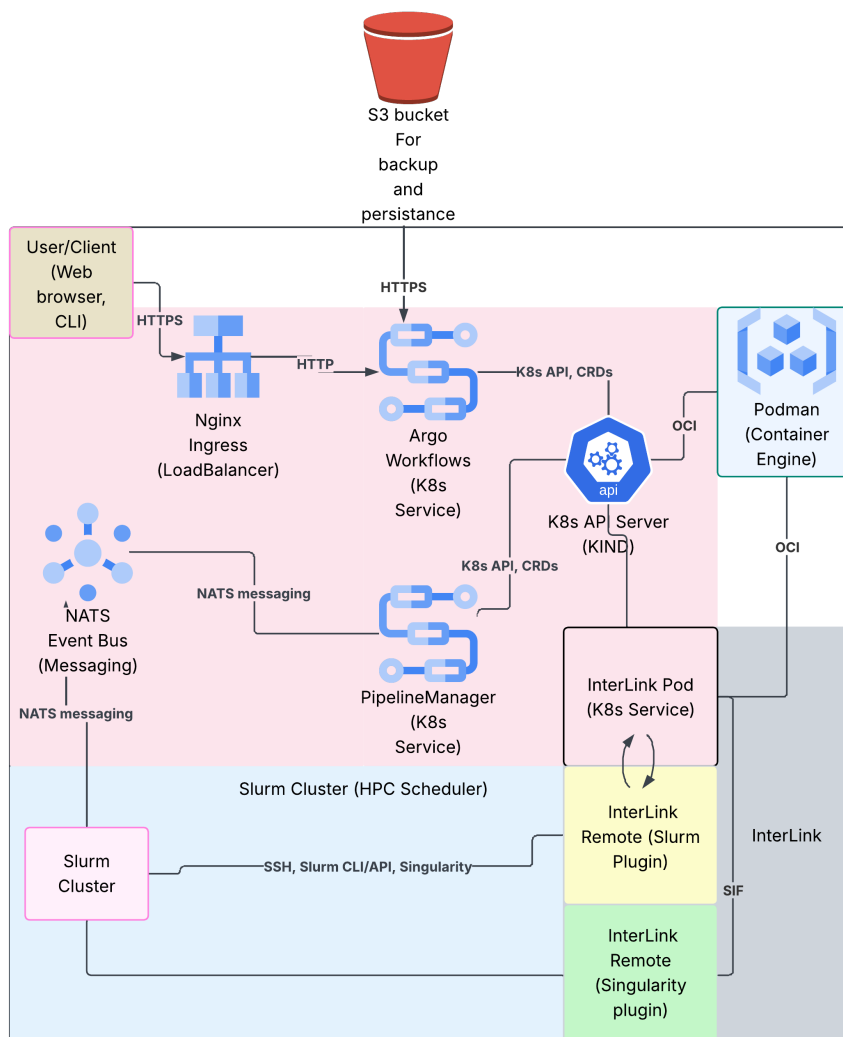


Figure 6.3: DCC Cloud Architecture German Solution

The prerequisite analysis revealed multiple categories of requirements spanning infrastructure, authentication, networking, and software dependencies. Infrastructure requirements include access to cloud computing resources with specific permissions for container orchestration, storage systems with S3 compatibility, and networking configurations that support secure inter-site communication. Authentication prerequisites involve credentials for multiple systems, including CNES Artifactory for container image access, ESA GitLab for source code repositories, and Google Cloud Platform for storage and compute resources when needed for federated operations. This is true for all types of deployment of the DDPC, but it is not necessarily specific to the German AEI.

The software dependency analysis specific to the German solution highlighted the critical importance of certain container technologies in the deployment process. The German implementation relies heavily on Podman as a rootless container engine that can operate within HPC security constraints while still providing OCI-compatible functionality required by Kubernetes. The

deployment tools consist of sophisticated bash scripts and Helm charts that automate the installation and configuration of Kind clusters, InterLink components, Argo Workflows, and supporting infrastructure. The InterLink component is the crucial bridge that translates Kubernetes pod specifications into Slurm job submissions, while maintaining the workflow orchestration capabilities provided by Argo Workflows. The German team's use of Podman as a rootless container engine enables the deployment of kind clusters without requiring administrative privileges or compromising system security.

6.4 Attempted Solutions

The deployment attempt began with a systematic approach to understanding and adapting the German AEI bridge solution to Deucalion's specific environment. The German implementation provided a valuable reference point because their HPC environment shares characteristics similar to Deucalion's, particularly the reliance on Slurm for job scheduling and the need to bridge Kubernetes orchestration with traditional HPC infrastructure. Their solution demonstrated that successful DDPC integration was possible in HPC environments through careful architectural adaptation and the strategic use of bridging technologies.

The initial phase involved a detailed analysis of the German deployment scripts and documentation to understand the specific dependencies and configuration requirements. This analysis revealed that their solution successfully addresses many challenges inherent in deploying cloud-native orchestration technologies on traditional HPC infrastructure. However, the fundamental barrier became apparent during the attempts to install the required container engine dependencies. The German Solution's reliance on Podman stems from the kind's absolute requirement for an OCI-compatible container engine to create and manage Kubernetes node containers. Despite its excellent capabilities for running scientific workloads in HPC environments, Singularity cannot fulfill this role because it lacks the Docker-compatible API required for cluster management. The architectural incompatibility extends beyond simple tool substitution – kind fundamentally expects to interact with a container engine that supports Docker-style image building, loading, and container lifecycle management, capabilities that Singularity does not provide in a compatible manner.

The investigation into potential workarounds revealed that existing solutions for Singularity-Kubernetes integration, such as Singularity-CRI, target different use cases and do not address the specific requirements of kind-based cluster deployment. These solutions focus on enabling Kubernetes to run Singularity containers on worker nodes, rather than allowing the deployment of Kubernetes control plane components using Singularity as the container engine. The fundamental architectural mismatch between Singularity's design philosophy and the requirements of Kubernetes cluster management tools represents a significant technical challenge that cannot be resolved through configuration changes or minor adaptations.

6.5 Proposed Solutions and Recommendations

Given the technical constraints identified during the deployment attempts, several potential solutions emerge that could enable successful DDPC integration with Deucalion. These approaches range from straightforward infrastructure modifications to fundamental architectural redesigns that address the core compatibility challenges between traditional HPC environments and cloud-native orchestration systems. Each solution presents different trade-offs regarding implementation complexity, security implications, development effort, and long-term sustainability within the European HPC ecosystem.

Specifically, there should be two deployment types because the components that could not be deployed are components responsible for the orchestration, and we only need to be a DCC. We can also conclude that we, as a DCC, can function normally because success was seen in the previous chapter.

The investigation has revealed that resolving the deployment challenges requires careful consideration of multiple factors, including administrative feasibility, resource requirements, and alignment with existing workflows. The solutions vary significantly in their approach to bridging the technological gap between Singularity-based containerization and Kubernetes-dependent orchestration systems, necessitating a thorough evaluation of each option's viability within Portugal's national computing infrastructure constraints. The collaborative engagement with the Deucalion technical team has also highlighted innovative approaches that could simplify the integration process while maintaining the essential functionality of the LISA DDPC network.

Table 6.1: Proposed Solutions for DDPC Integration with Deucalion

Proposed Solution	Pros	Cons
Install Podman alongside Singularity	• Straightforward implementation approach • Maintains existing Singularity workflows • Enables kind-based Kubernetes deployment • Minimal changes to DDPC framework architecture • Leverages proven German AEI solution	• Requires administrative approval and system modifications • Potential security and stability implications • Unknown impact on existing computational workloads • Kernel feature requirements need assessment • May introduce additional maintenance overhead

Proposed Solution	Pros	Cons
Modified DDPC deployment without kind dependency	• Eliminates container engine compatibility issues • Works within existing Deucalion constraints • Maintains DDPC network compatibility • Custom solution tailored for HPC environments • Preserves the security model of traditional HPC	• Requires significant development effort • Needs close collaboration with LISA development teams • Potential incompatibility with future DDPC updates • Complex maintenance and long-term support requirements • Risk of diverging from standard DDPC architecture
Hybrid cloud-HPC orchestration model	• Minimal infrastructure changes on Deucalion • Leverages cloud scalability for the orchestration layer • Utilizes HPC computational efficiency for workloads • Primarily requires network-related modifications • Enables participation with restrictive HPC policies	• Complex network security considerations • Data movement and latency performance implications • Dependency on external cloud infrastructure • Potential increased operational complexity • May require specialized network configurations
Enhanced container engine interoperability standards	• Benefits the entire scientific computing community • Provides a long-term, sustainable solution • Addresses fundamental compatibility issues • Potential for EuroHPC-wide standardization • Aligns with broader HPC evolution trends	• Extended development and standardization timeline • Requires broad community consensus and adoption • Uncertain implementation schedule • No immediate solution for current LISA requirements • May face resistance from existing HPC practices

Proposed Solution	Pros	Cons
Script-based remote execution model	• Completely bypasses container orchestration issues • Utilizes existing SSH infrastructure capabilities • Simple and reliable implementation approach • Democratizes participation for smaller HPC centers • Reduces technical barriers for integration	• Represents a paradigm shift from a cloud-native approach • Potentially reduced workflow orchestration capabilities • Security considerations with remote SSH access • May limit access to advanced workflow features • Could impact real-time monitoring and control

Long-term solutions might involve advocacy within the broader HPC community for enhanced container engine interoperability standards, enabling tools like kind to operate with Singularity-based runtimes. The growing adoption of Kubernetes in HPC environments suggests that standardization efforts could benefit the entire scientific computing community. Collaboration with EuroHPC initiatives could also influence future supercomputer procurement and configuration decisions, including support for multiple container engine technologies.

6.5.1 Presentation to Deucalion Team

The findings and challenges encountered during the deployment attempts were presented to the Deucalion technical team to facilitate collaborative problem-solving and explore potential paths forward. The presentation emphasized the scientific importance of LISA participation for Portuguese astrophysics research and the significant investment already made in developing Portugal's contribution to the DDPC network. The technical challenges were presented not as barriers, but as engineering problems that could be addressed through collaborative solutions.

During the discussions with the Deucalion team, it became evident that, given the core components of the DDPC architecture, we could not deploy it as a fundamentally cloud-native application; a more practical approach would be to shift the paradigm entirely. The team suggested operating the orchestration layer from a central station—outside of Deucalion itself—which would remotely initiate computation on the supercomputer via SSH. In this model, the central station would execute scripts that connect to Deucalion, fetch the necessary data, run the required computations, and then retrieve the results. This approach bypasses the need for complete cloud-native infrastructure within the HPC environment, making the solution significantly more feasible for all participating countries, especially those with smaller or more restrictive HPC centers. By decoupling orchestration from the HPC and leveraging secure, script-based remote execution, this model could democratize participation in the LISA DDPC network and lower the technical barriers for integration across diverse European infrastructures.

Chapter 7

Conclusion

This thesis investigated the deployment of the LISA GlobalFit distributed computing architecture on Portugal’s national High-Performance Computing infrastructure, specifically the Deucalion supercomputer. The research addressed the critical challenge of integrating cloud-native scientific workflow orchestration systems with traditional HPC environments. This problem extends beyond the LISA mission to affect the broader scientific computing community.

The work demonstrated that Portugal has the technical capabilities to contribute meaningfully to the LISA Distributed Data Processing Centre (DDPC) network by successfully deploying and validating the isolated GlobalFit framework on Deucalion. However, the research also revealed fundamental architectural incompatibilities between traditional HPC containerization technologies and modern cloud-native orchestration requirements, highlighting significant challenges in the evolution of scientific computing infrastructure.

The investigation provided valuable insights into the complexities of bridging different technological paradigms while establishing a foundation for future integration efforts. Although a complete DCC architecture deployment was not achieved, the research outcomes contribute significantly to understanding the requirements and constraints of deploying advanced scientific workflow orchestration systems on traditional HPC infrastructure.

7.1 Summary of Findings

The research achieved several significant technical milestones, demonstrating Portugal’s readiness to participate in the LISA mission. The isolated GlobalFit framework was successfully deployed, configured, and validated locally and on Deucalion’s ARM partition, representing the first successful implementation of this sophisticated gravitational wave data analysis system on the Portuguese national computing infrastructure.

Performance validation confirmed that the framework effectively utilized all available computational resources, achieving 100% CPU utilization during execution and successfully scaling from simple test cases involving single Gaussian fitting to complex scenarios with thousands of sources

and samples. The framework's critical checkpoint and resume functionality was validated, ensuring robustness during system interruptions or job scheduling constraints essential for long-running HPC computations.

Critical operational features essential for long-running HPC computations were thoroughly validated:

- Checkpoint and resume functionality: Ensuring robustness during system interruptions or job scheduling constraints
- Multi-architecture compatibility: Successful adaptation to Deucalion's heterogeneous ARM-based environment
- Resource management integration: Seamless operation within Slurm's job scheduling framework
- Scalability validation: Confirmed capability to handle datasets ranging from simple test cases to complex multi-source scenarios

7.1.1 Infrastructure Analysis and Containerization Challenges

The comprehensive analysis of Deucalion's infrastructure revealed strengths and critical limitations directly impacting DCC deployment capabilities. While Singularity provides excellent support for scientific computing workloads with superior security and performance characteristics, its architectural incompatibility with Kubernetes-based orchestration systems creates insurmountable barriers for deploying cloud-native workflow management frameworks.

The investigation into containerization technologies revealed fundamental philosophical differences between HPC-oriented and cloud-native approaches:

- Singularity strengths: Designed specifically for HPC environments with excellent security, performance, and scientific workflow integration
- Kubernetes requirements: Demands OCI-compatible container engines that are architecturally incompatible with Singularity's design
- Bridge technologies: Existing solutions like Singularity-CRI target different use cases and cannot address kind-based cluster deployment requirements

7.1.2 DDPC Deployment Challenges and Architectural Incompatibilities

The attempt to deploy the complete DCC architecture revealed that the root cause of the deployment failure lies in the architectural incompatibility between Singularity container technology, which excels in HPC scientific computing environments, and kind-based Kubernetes deployment tools that require OCI-compatible container engines like Docker or Podman. This incompatibility is not merely a configuration issue but represents a fundamental architectural mismatch that cannot

be resolved through minor adaptations or workarounds. The successful German AEI bridge implementation demonstrates that DDPC integration is achievable in HPC environments, but requires specific container technology capabilities that are not currently available on Deucalion.

Despite extensive investigation of potential workarounds and alternative deployment strategies, no viable solution could bridge this technological gap without requiring significant infrastructure modifications or compromising system security and stability.

Analysis of the German solution revealed key components that enabled their success:

- Podman integration: Rootless container engine providing OCI compatibility within HPC security constraints
- InterLink technology: Custom bridging solution translating Kubernetes pod specifications to Slurm job submissions
- Hybrid architecture: Cloud-based orchestration layer interfacing with traditional HPC compute resources

Since the parts that could not be deployed correspond to the orchestrator, and we only need to be a DCC and not the central orchestrator, we, as a part of Deucalion, can function as usual despite these issues.

7.2 Contributions

This research makes several critical technical contributions to the scientific computing community. The successful deployment methodology developed for the GlobalFit framework on heterogeneous HPC infrastructure provides a replicable approach that can benefit other European HPC centers seeking to participate in the LISA mission or deploy similar scientific computing frameworks.

The comprehensive documentation created for the GlobalFit framework addresses a significant gap in available resources, providing detailed architectural diagrams, communication flow descriptions, and component interaction specifications that were previously unavailable. This documentation serves both immediate implementation needs and longer-term research objectives. We want to point out that, when writing this thesis, the comprehensive documentation created was a necessary step to understand and enable us to deploy and test. But as of 4 days before the end, the GlobalFit APC framework was made public with the appropriate documentation. While we successfully validated our efforts against this documentation, it did not arrive meaningfully in time to steer or assist this thesis meaningfully. The framework is available in the following link <https://gitlab.in2p3.fr/lisa-apc/global-fit>

This thesis's key technical contributions include:

- Multi-architecture deployment strategies: Demonstrating Python-based scientific framework adaptation across ARM and x86 processor architectures
- Containerization best practices: Developing HPC-specific approaches for managing containerized scientific workflows

- Performance optimization techniques: Validating resource utilization strategies for compute-intensive astrophysical applications
- Integration methodologies: Creating systematic approaches for bridging traditional HPC and modern workflow orchestration systems

7.3 Limitations

The primary limitation encountered in this research stems from fundamental infrastructure constraints that prevent the deployment of the complete DDPC architecture on Deucalion. The absence of Docker or Podman container engines, while justifiable from security and stability perspectives in HPC environments, creates an insurmountable barrier for deploying Kubernetes-based orchestration systems that form the foundation of the DDPC architecture.

These constraints are not unique to Deucalion but reflect broader tensions between traditional HPC security models and the requirements of modern cloud-native computing paradigms. The research revealed that resolving these constraints requires significant infrastructure modifications or fundamental architectural changes to the DDPC deployment strategy, representing a systemic challenge rather than a local implementation issue.

Technical limitations identified include:

- Container engine incompatibility: Fundamental architectural mismatch between Singularity and Kubernetes requirements
- Security model conflicts: Traditional HPC security approaches limit cloud-native technology adoption
- Administrative constraints: Policy and governance limitations on infrastructure modifications
- Technology maturity gaps: Limited availability of mature bridging solutions for HPC-cloud integration

The limited availability of comprehensive documentation for the GlobalFit framework initially constrained the research progress, requiring significant effort to reverse-engineer system behaviors and create the necessary technical understanding for successful deployment.

The research focused specifically on the Portuguese national HPC infrastructure and the French version of the GlobalFit framework that LISA will use, and where all the others that may be created will be based on this one, which may limit the generalizability of findings to other HPC environments or framework implementations. While the challenges identified likely represent broader compatibility issues, validation across additional systems would strengthen the conclusions.

The research was also limited by the prototype nature of the GlobalFit framework, which is still under active development. Some limitations encountered may be addressed in future framework versions, potentially affecting the long-term relevance of specific technical findings.

7.4 Future Work

Future research should prioritize the development of enhanced interoperability solutions that bridge the technological gap between traditional HPC container technologies and modern cloud-native orchestration tools. This represents a critical area where targeted development efforts could benefit the broader scientific computing community significantly.

The most promising immediate research directions include developing enhanced InterLink capabilities with reduced Kubernetes infrastructure requirements, enabling direct integration between Argo Workflows and Slurm scheduling systems without requiring full Kubernetes cluster deployment on HPC resources. This approach would require significant software engineering effort but could provide benefits beyond the LISA use case by enabling broader adoption of cloud-native workflow orchestration in traditional HPC environments.

Priority development areas include:

- Simplified orchestration architectures: Reducing Kubernetes dependencies while maintaining workflow management capabilities
- Enhanced bridging technologies: Developing more sophisticated InterLink-style solutions for HPC-cloud integration
- Container engine interoperability: Creating standards and tools for cross-platform container technology compatibility
- Performance optimization frameworks: Developing systematic approaches for optimizing containerized scientific workflows on HPC systems

Collaborative solutions with the Deucalion technical team and the broader EuroHPC community should be actively pursued to address the fundamental container engine compatibility challenges. Future work should explore the feasibility of installing Podman as an additional container engine alongside existing Singularity installations, carefully evaluating security implications, kernel feature requirements, and potential impacts on existing workloads.

The hybrid deployment model suggested by the Deucalion team represents a particularly promising avenue for future development. This approach would operate orchestration components on external cloud infrastructure while directly integrating Deucalion's Slurm scheduling system through enhanced remote execution capabilities. Such a model could significantly reduce the technical barriers for HPC integration while maintaining the security and performance requirements for shared national computing resources.

Strategic development priorities include:

- Multi-container engine support: Investigating technical and policy pathways for supporting multiple containerization technologies
- Hybrid cloud-HPC architectures: Developing frameworks for distributed orchestration across cloud and HPC environments

- Security model evolution: Creating approaches that balance traditional HPC security with cloud-native flexibility requirements
- International coordination mechanisms: Establishing collaborative frameworks for addressing shared technical challenges across EuroHPC sites

7.5 Concluding Remarks

The successful deployment of the isolated GlobalFit framework on Deucalion establishes a solid foundation for Portugal's participation in the LISA mission. It demonstrates the country's technical readiness for international scientific collaborations of unprecedented scale and complexity. The lessons learned from this deployment attempt provide valuable insights that could accelerate the adoption of advanced workflow orchestration technologies across the EuroHPC network, ultimately enhancing Europe's capacity for cutting-edge scientific research and international collaboration.

This research contributes to the broader understanding of how scientific computing infrastructure must evolve to meet the demands of next-generation astrophysical missions and international research collaborations.

7.6 Alignment with UN Sustainable Development Goals

Beyond the immediate technical and scientific contributions documented in this thesis, this research makes meaningful contributions to the United Nations Sustainable Development Goals (SDGs), demonstrating how advanced scientific computing research can serve broader societal objectives.

The work primarily contributes to **SDG 4 (Quality Education)** by developing comprehensive technical documentation and educational resources that advance scientific computing capabilities. The systematic approach to deploying distributed computing frameworks provides valuable learning materials for future researchers and contributes to building technical capacity in emerging technologies essential for scientific research.

Significant contributions to **SDG 9 (Industry, Innovation and Infrastructure)** are evident through the successful deployment of resilient computing infrastructure and the development of innovative solutions for integrating cloud-native orchestration with traditional HPC environments. This work strengthens the country's research infrastructure and technological capabilities by establishing Portugal's ability to participate in international scientific collaborations requiring advanced computational resources.

The research exemplifies **SDG 17 (Partnerships for the Goals)** through Portugal's successful integration into the international LISA consortium and broader European high-performance computing ecosystem. This work demonstrates how scientific research can catalyze partnerships

across multiple sectors and geographical boundaries, facilitating knowledge sharing and technological cooperation in pursuit of advancing gravitational wave astronomy. This alignment with the SDGs highlights the interconnected nature of scientific research, technological development, and sustainable development, demonstrating how fundamental physics research through advanced computing technologies and international collaboration supports the broader vision of science and technology as drivers of sustainable development outlined in the 2030 Agenda.

References

- [1] Damir Regvart (CARNET) et al. *Data Transfer Node (DTN) Tests on the GÉANT Testbeds Service (GTS)*. Tech. rep. GÉANT Association, Dec. 2020. URL: https://resources.geant.org/wp-content/uploads/2022/02/GN4-3_White-Paper_DTN.pdf.
- [2] Subil Abraham. “The HPC Container Experience on the Summit Supercomputer”. In: *Practice and Experience in Advanced Research Computing*. ACM, July 2023, pp. 273–277. ISBN: 9781450399852. DOI: 10.1145/3569951.3597608.
- [3] National Network for Advanced Computing. *DEUCALION*. June 2025. URL: <https://rnca.fccn.pt/en/deucalion/#:~:text=One%20of%20the%20first%20,financed%20by%20EuroHPC>.
- [4] Akash. *Docker vs Podman: An In-Depth Comparison (2025)*. June 2025. URL: https://dev.to/mechcloud_academy/docker-vs-podman-an-in-depth-comparison-2025-2eia.
- [5] Osamah I. Alqaisi, Ali Şaman Tosun, and Turgay Korkmaz. “Performance Analysis of Container Technologies for Computer Vision Applications on Edge Devices”. In: *IEEE Access* 12 (2024), pp. 41852–41869. ISSN: 2169-3536. DOI: 10.1109/ACCESS.2024.3376570.
- [6] The Kubernetes Authors. “Production-Grade Container Orchestration”. In: (Dec. 2024). URL: <https://kubernetes.io/?spm=a2c65.11461447.0.0.15d87863YiFv0v>.
- [7] Stanislav Babak et al. “Science with the space-based interferometer LISA. V. Extreme mass-ratio inspirals”. In: *Physical Review D* 95 (10 May 2017), p. 103012. ISSN: 2470-0010. DOI: 10.1103/PhysRevD.95.103012.
- [8] Susanne M. Balle and Daniel J. Palermo. “Enhancing an Open Source Resource Manager with Multi-core/Multi-threaded Support”. In: *Job Scheduling Strategies for Parallel Processing*. Springer Berlin Heidelberg, pp. 37–50. DOI: 10.1007/978-3-540-78699-3_3.
- [9] Antoine Basset. *GlobalFit Framework, Internal ESA Document*. June 2024.
- [10] Ugo Becciani. “ASTROPHYSICS & COSMOS OBSERVATIONS”. In: (Dec. 2024). URL: <https://www.supercomputing-isc.it/en/spoke-3-astrophysics-cosmos-observations-en/>.
- [11] Angel M. Beltre et al. “Enabling HPC Workloads on Cloud Infrastructure Using Kubernetes Container Orchestration Mechanisms”. In: *2019 IEEE/ACM International Workshop on Containers and New Orchestration Paradigms for Isolated Environments in HPC (CANOPIE-HPC)*. IEEE, Nov. 2019, pp. 11–20. ISBN: 978-1-7281-6028-3. DOI: 10.1109/CANOPIE-HPC49598.2019.00007.
- [12] P Bender et al. “LISA: a cornerstone mission for the observation of gravitational waves”. In: *System and technology study report ESA-SCI 11* (2000), p. 2000.

- [13] Peter L Bender. “Expected LISA mission performance”. In: *European Space Agency, (Special Publication) ESA SP*. Cited by: 0. 1997, pp. 259–263. URL: <https://www.scopus.com/inward/record.uri?eid=2-s2.0-5244282341&partnerID=40&md5=5bbe22c5e135fee4ed5158c492bb6293>.
- [14] David Bernstein. “Containers and Cloud: From LXC to Docker to Kubernetes”. In: *IEEE Cloud Computing* 1 (3 Sept. 2014), pp. 81–84. ISSN: 2325-6095. DOI: 10.1109/MCC.2014.51.
- [15] Carlos Boneti et al. “Balancing HPC applications through smart allocation of resources in MT processors”. In: *2008 IEEE International Symposium on Parallel and Distributed Processing*. IEEE, Apr. 2008, pp. 1–12. ISBN: 978-1-4244-1693-6. DOI: 10.1109/IPDPS.2008.4536293.
- [16] Téo Bouvard. *LISA GlobalFit Framework Architecture*. Nov. 2024.
- [17] Joe Brockmeier. *Open Container Initiative specifications reach 1.0*. July 2017. URL: <https://www.redhat.com/en/blog/open-container-initiative-specifications-reach-10>.
- [18] Pawsey Supercomputing Centre. “Basics of Singularity”. In: (June 2025). URL: <https://pawsey.github.io/hpc-container-training/13-singularity-intro/index.html>.
- [19] Joana Chavez. *Singularity : a "Docker" for HPC environments*. June 2018. URL: <https://dev.to/grokcode/singularity--a-docker-for-hpc-environments-i6p>.
- [20] Nelson Christensen and Renate Meyer. “Using Markov chain Monte Carlo methods for estimating parameters with gravitational radiation data”. In: *Physical Review D* 64 (2 May 2001), p. 022001. ISSN: 0556-2821. DOI: 10.1103/PhysRevD.64.022001.
- [21] EU Commission. *Deucalion: New EuroHPC supercomputer inaugurated in Portugal*. Sept. 2023. URL: https://ieu-monitoring.com/editorial/deucalion-new-eurohpc-supercomputer-inaugurated-in-portugal/415219?utm_source=ieu-portal#:~:text=Deucalion%20the%20seventh%20EuroHPC%20supercomputer,available%20by%20the%20EuroHPC%20systems.
- [22] Karsten Danzmann and the LISA study team. “LISA: laser interferometer space antenna for gravitational wave measurements”. In: *Classical and Quantum Gravity* 13 (11A Nov. 1996), A247–A250. ISSN: 0264-9381. DOI: 10.1088/0264-9381/13/11A/033.
- [23] Senwen Deng et al. “Modular global-fit pipeline for LISA data analysis”. In: (Apr. 2025). DOI: 10.48550/arXiv.2501.10277.
- [24] Manoj Himmatrao Devare. *Challenges and Opportunities in High Performance Cloud Computing*. Cited by: 4. 2021, pp. 1989–2018. DOI: 10.4018/978-1-7998-5339-8.ch096. URL: <https://www.scopus.com/inward/record.uri?eid=2-s2.0-85111071605&doi=10.4018%2f978-1-7998-5339-8.ch096&partnerID=40&md5=66fda83eebe85ca261835f8026206744>.
- [25] DODHPC. “Introduction to Singularity Containers on HPC”. In: (June 2025). URL: <https://centers.hpc.mil/users/docs/general/singularity.html>.
- [26] J. Dongarra et al. “High-performance computing: clusters, constellations, MPPs, and future directions”. In: *Computing in Science & Engineering* 7 (2 Mar. 2005), pp. 51–59. ISSN: 1521-9615. DOI: 10.1109/MCSE.2005.34.

- [27] Borislav Dordevic et al. “Performance comparison of Docker and Podman container-based virtualization”. In: *2022 21st International Symposium INFOTEH-JAHORINA (INFOTEH)*. IEEE, Mar. 2022, pp. 1–6. ISBN: 978-1-6654-3778-3. DOI: 10.1109/INFOTEH53737.2022.9751277.
- [28] Pierre Dubath et al. “The Euclid Data Processing Challenges”. In: (Jan. 2017). DOI: 10.1017/S1743921317001521.
- [29] EU. *EUROHPC DEUCALION*. Jan. 2025. URL: https://eurohpc-ju.europa.eu/deucalion-new-eurohpc-world-class-green-supercomputer-portugal-2021-02-05_en.
- [30] eu. *EUROHPC*. June 2025. URL: <https://eurohpc-ju.europa.eu/system/files/2023-07/EuroHPC%20Systems%20Report-Sep2021.pdf>.
- [31] EuroHPC/2024/OP/0007. *ACQUISITION, DELIVERY, INSTALLATION AND HARDWARE AND SOFTWARE MAINTENANCE OF THE UPGRADE OF THE EUROHPC SUPER-COMPUTER LEONARDO – LISA*. Sept. 2024. URL: https://ec.europa.eu/info/funding-tenders/opportunities/portal/screen/opportunities/tender-details/docs/98cd4117-4277-4803-84e4-fae6c38afb49-CN/LISA%20upgrade-technical-specifications_V1.pdf.
- [32] fccn. *ECA*. Nov. 2024. URL: <https://indico.fccn.pt/event/41/contributions/798/attachments/604/1146/ASousa-RNCA2024.pdf>.
- [33] The Linux Foundation. *Open Container Initiative (OCI) Releases v1.0 of Container Standards*. July 2017. URL: <https://www.linuxfoundation.org/press/press-release/open-container-initiative-oci-releases-v1-0-of-container-standards>.
- [34] Nishanth Dandapanthula Rengan Xu Frank Han. “Containerizing HPC Applications with Singularity”. In: (Sept. 2017). URL: https://dl.dell.com/manuals/all-products/esuprt_solutions_int/esuprt_solutions_int_solutions_resources/high-computing-solution-resources_white-papers10_en-us.pdf.
- [35] Yihan Fu et al. “Probabilistic Compute-in-Memory Design for Efficient Markov Chain Monte Carlo Sampling”. In: *IEEE Transactions on Circuits and Systems I: Regular Papers* 71 (2 Feb. 2024), pp. 703–716. ISSN: 1558-0806. DOI: 10.1109/tcsi.2023.3337529. URL: <http://dx.doi.org/10.1109/TCSI.2023.3337529>.
- [36] Peter F Gath et al. “LISA mission and system architectures and performances”. In: *Journal of Physics: Conference Series*. Vol. 154. Cited by: 7; All Open Access, Gold Open Access. 2009. DOI: 10.1088/1742-6596/154/1/012013. URL: <https://www.scopus.com/inward/record.uri?eid=2-s2.0-66149126078&doi=10.1088%2f1742-6596%2f154%2f1%2f012013&partnerID=40&md5=e6816ebd43ca150f4d03206030acf596>.
- [37] Keole Gazoline. *Container Technology for HPC*. Dec. 2024. URL: <https://www.doitnowgroup.com/technical-papers/container-technology-for-hpc/>.
- [38] Maxime Gonthier et al. “Data-Driven Locality-Aware Batch Scheduling”. In: *2024 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*. IEEE, May 2024, pp. 202–211. ISBN: 979-8-3503-6460-6. DOI: 10.1109/IPDPSW63119.2024.00058.

- [39] Daniel Gruber et al. *Kubernetes, Containers and HPC*. Sept. 2019. URL: <https://www.hpcwire.com/2019/09/19/kubernetes-containers-and-hpc/>.
- [40] Cameron Hashemi-Pour. *What is Docker? And how it works?* Dec. 2024. URL: <https://www.techtarget.com/searchitoperations/definition/Docker>.
- [41] Red Hat. *What is Podman?* June 2024. URL: <https://www.redhat.com/en/topics/containers/what-is-podman>.
- [42] Husain Husain et al. "Analysis and Implementation of Comparison Between Podman and Docker in Container Management". In: *International Journal of Electronics and Communications Systems* 3 (2 Dec. 2023), pp. 57–67. ISSN: 2798-2610. DOI: 10.24042/ijecs.v3i2.19860.
- [43] Shubham Jasani. "Docker vs. Singularity: Which Fits Your Workflow Better?" In: (June 2025). URL: <https://www.dhiwise.com/post/docker-vs-singularity-which-fits-your-workflow-better>.
- [44] Seung-Seop Jin, Heekun Ju, and Hyung-Jo Jung. "Adaptive Markov chain Monte Carlo algorithms for Bayesian inference: recent advances and comparative study". In: *Structure and Infrastructure Engineering* 15 (11 Nov. 2019), pp. 1548–1565. ISSN: 1573-2479. DOI: 10.1080/15732479.2019.1628077.
- [45] Nikolaos Karnesis et al. "Eryn: a multipurpose sampler for Bayesian inference". In: *Monthly Notices of the Royal Astronomical Society* 526 (4 Oct. 2023), pp. 4814–4830. ISSN: 0035-8711. DOI: 10.1093/mnras/stad2939. URL: <https://academic.oup.com/mnras/article/526/4/4814/7284403>.
- [46] Michael L Katz et al. *An efficient GPU-accelerated multi-source global fit pipeline for LISA data analysis*. 2024. URL: <https://arxiv.org/abs/2405.04690>.
- [47] Michael L. Katz et al. "Assessing the data-analysis impact of LISA orbit approximations using a GPU-accelerated response model". In: *Physical Review D* 106 (10 Nov. 2022), p. 103001. ISSN: 2470-0010. DOI: 10.1103/PhysRevD.106.103001.
- [48] Michael L. Katz et al. "GPU-accelerated massive black hole binary parameter estimation with LISA". In: *Physical Review D* 102 (2 July 2020), p. 023033. ISSN: 2470-0010. DOI: 10.1103/PhysRevD.102.023033.
- [49] Ritva A. Keski-Kuha et al. "LISA telescope development status and flight design". In: *Space Telescopes and Instrumentation 2024: Optical, Infrared, and Millimeter Wave*. Ed. by Laura E. Coyle, Marshall D. Perrin, and Shuji Matsuura. Vol. 13092. Cited by: 0. SPIE, Aug. 2024, p. 88. ISBN: 9781510675070. DOI: 10.1117/12.3019158. URL: <https://www.spiedigitallibrary.org/conference-proceedings-of-spie/13092/3019158/LISA-telescope-development-status-and-flight-design/10.1117/12.3019158.full>.
- [50] Kong. *What is Kubernetes? A Comprehensive Guide to Container Orchestration*. Dec. 2024. URL: <https://konghq.com/blog/learning-center/what-is-kubernetes>.
- [51] Mandeep Kumar and Gagandeep Kaur. "Study of container-based JupyterLab and AI Framework on HPC with GPU usage". In: *2022 International Conference on Smart Generation Computing, Communication and Networking (SMART GENCON)*. IEEE, Dec. 2022, pp. 1–5. ISBN: 978-1-6654-5499-5. DOI: 10.1109/SMARTGENCON56628.2022.10084107.
- [52] Robert Lalonde. *Kubernetes Meets High-Performance Computing*. Aug. 2017. URL: <https://kubernetes.io/blog/2017/08/kubernetes-meets-high-performance/>.

- [53] Guohua Li, Joon Woo, and Sang Boem Lim. “HPC Cloud Architecture to Reduce HPC Workflow Complexity in Containerized Environments”. In: *Applied Sciences* 11 (3 Jan. 2021), p. 923. ISSN: 2076-3417. DOI: [10.3390/app11030923](https://doi.org/10.3390/app11030923).
- [54] Tyson Littenberg and Neil Cornish. *Designing a Global Fit Analysis for LISA*. Aug. 2020. URL: https://www.youtube.com/watch?v=IsC58EJCL-E%5C&ab_channel=LISAMission.
- [55] Tyson B. Littenberg and Neil J. Cornish. “Prototype global analysis of LISA data with multiple source types”. In: *Physical Review D* 107 (6 Mar. 2023), p. 063004. ISSN: 2470-0010. DOI: [10.1103/PhysRevD.107.063004](https://doi.org/10.1103/PhysRevD.107.063004).
- [56] Ian Lumb. *THE SINGULARITY-KUBERNETES INTEGRATION: FROM A DEEP LEARNING USE CASE TO THE TECHNICAL SPECIFICS*. Apr. 2019. URL: <https://medium.com/sylabs/the-singularity-kubernetes-integration-from-a-deep-learning-use-case-to-the-technical-specifcs-927c22402068>.
- [57] MACC. *Deucalion User Guide*. June 2025. URL: <https://docs.macc.fccn.pt/deucalion/>.
- [58] Nick Martin. *Open Container Initiative*. May 2023. URL: <https://www.techtarget.com/searchitoperations/definition/Open-Container-Initiative>.
- [59] NERSC. *Job Scheduling*. June 2025. URL: <https://docs.nersc.gov/jobs/scheduling/>.
- [60] W. O’Mullane et al. “Gaia Data Processing Architecture”. In: (Nov. 2006). DOI: [10.48550/arXiv.astro-ph/0611885](https://doi.org/10.48550/arXiv.astro-ph/0611885).
- [61] openshift. *Podman*. June 2025. URL: <https://github.com/containers/podman>.
- [62] Jessica Page and Tyson B. Littenberg. “Bayesian time delay interferometry for orbiting LISA: Accounting for the time dependence of spacecraft separations”. In: *Physical Review D* 108 (4 Aug. 2023), p. 044065. ISSN: 2470-0010. DOI: [10.1103/PhysRevD.108.044065](https://doi.org/10.1103/PhysRevD.108.044065).
- [63] Hugo Palacin. *Architecture Poster, Internal ESA Document*. July 2024.
- [64] Jose Antonio Pascual, Javier Navaridas, and Jose Miguel-Alonso. “Effects of Topology-Aware Allocation Policies on Scheduling Performance”. In: 2009, pp. 138–156. DOI: [10.1007/978-3-642-04633-9_8](https://doi.org/10.1007/978-3-642-04633-9_8).
- [65] Fabio Pasian et al. “Science ground segment for the ESA Euclid Mission”. In: ed. by Nicole M. Radziwill and Gianluca Chiozzi. Sept. 2012, p. 845104. DOI: [10.1117/12.926026](https://doi.org/10.1117/12.926026).
- [66] Antoine Petiteau. “LISA Data Processing Group”. In: (Dec. 2024). URL: https://lisa.pages.in2p3.fr/consortium-userguide/g_ldpg.html.
- [67] Wilsa Azmalia Putri. *What is Kubernetes? Definition, Features, and How It Works*. Dec. 2024. URL: <https://www.centraldatatech.com/blog-news/what-is-kubernetes-definition-features-and-how-it-works/>.
- [68] Basit Qureshi. “Adaptive Multi-Criteria Selection for Efficient Resource Allocation in Frugal Heterogeneous Hadoop Clusters”. In: *Electronics* 13 (10 May 2024), p. 1836. ISSN: 2079-9292. DOI: [10.3390/electronics13101836](https://doi.org/10.3390/electronics13101836).
- [69] Omer Rathore et al. “Load Balancing For High Performance Computing Using Quantum Annealing”. In: (Mar. 2024). DOI: [10.48550/arXiv.2403.05278](https://doi.org/10.48550/arXiv.2403.05278).
- [70] Jake Roach. “Docker vs. Podman: Which Containerization Tool is Right for You”. In: (June 2025). URL: <https://www.datacamp.com/blog/docker-vs-podman>.

- [71] Iuri La Rosa et al. “Continuous Gravitational-Wave Data Analysis with General Purpose Computing on Graphic Processing Units”. In: *Universe* 7 (7 June 2021), p. 218. ISSN: 2218-1997. DOI: [10.3390/universe7070218](https://doi.org/10.3390/universe7070218).
- [72] Isabel Escudero Sanz, Astrid Heske, and Jeffrey C. Livas. “A telescope for LISA – the Laser Interferometer Space Antenna”. In: *Advanced Optical Technologies* 7 (6 Dec. 2018), pp. 395–400. ISSN: 2192-8584. DOI: [10.1515/aot-2018-0044](https://doi.org/10.1515/aot-2018-0044).
- [73] schedmd. “Slurm”. In: (June 2025). URL: <https://slurm.schedmd.com/overview.html>.
- [74] Aqua Security. “VM vs. Container: Ultimate 2024 Showdown”. In: (Dec. 2024). URL: <https://www.aquasec.com/cloud-native-academy/docker-container/vm-vs-container/>.
- [75] *Slurm Workload Manager*. June 2025. URL: https://en.wikipedia.org/wiki/Slurm_Workload_Manager.
- [76] Vanessa Sochat and Alec Scott. “Collaborative Container Modules with Singularity Registry HPC”. In: *Journal of Open Source Software* 6 (63 July 2021), p. 3311. ISSN: 2475-9066. DOI: [10.21105/joss.03311](https://doi.org/10.21105/joss.03311).
- [77] Stefan H. Strub et al. “Global Analysis of LISA Data with Galactic Binaries and Massive Black Hole Binaries”. In: (Mar. 2024). DOI: <https://doi.org/10.48550/arXiv.2403.15318>.
- [78] suse. *Running Podman in Rootless Mode*. June 2025. URL: <https://documentation.suse.com/smart/container/html/rootless-podman/index.html>.
- [79] Stephanie Susnjara. *What is container orchestration?* Dec. 2024. URL: <https://www.ibm.com/think/topics/container-orchestration>.
- [80] sylabs. *Introduction to Singularity*. June 2025. URL: <https://docs.sylabs.io/guides/3.5/user-guide/introduction.html>.
- [81] Giuliano Taffoni et al. “Redesign of astrophysical codes for exascale computing: the SPACE experience”. In: *Software and Cyberinfrastructure for Astronomy VIII*. Ed. by Gianluca Chiozzi and Jorge Ibsen. SPIE, July 2024, p. 32. ISBN: 9781510675254. DOI: [10.1117/12.3020361](https://doi.org/10.1117/12.3020361).
- [82] Open Institute of Technology. *Distributed Computing: Unraveling the Power of Parallelism & Cloud Systems*. July 2023. URL: <https://www.opit.com/magazine/distributed-computing/>.
- [83] world wide technologie. “Workload Management & Orchestration Series: Slurm Workload Manager”. In: (June 2025). URL: <https://www.wwt.com/blog/workload-management-and-orchestration-series-slurm-workload-manager>.
- [84] Tigera. “Key Components of Container Security Architecture”. In: (Dec. 2024). URL: <https://www.tigera.io/learn/guides/container-security-best-practices/>.
- [85] European High-Performance Computing Joint Undertaking. “NCC Portugal”. In: (Dec. 2024). URL: <https://www.eurocc-access.eu/about-us/meet-the-nccs/ncc-portugal/>.
- [86] Peter Vaillancourt et al. “Reproducible and Portable Workflows for Scientific Computing and HPC in the Cloud”. In: (June 2020). DOI: [10.1145/3311790.3396659](https://doi.org/10.1145/3311790.3396659).

- [87] Michele Vallisneri, Jeff Crowder, and Massimo Tinto. “Sensitivity and parameter-estimation precision for alternate LISA configurations”. In: *Classical and Quantum Gravity* 25 (6 Mar. 2008), p. 065005. ISSN: 0264-9381. DOI: [10.1088/0264-9381/25/6/065005](https://doi.org/10.1088/0264-9381/25/6/065005).
- [88] Eleni Vlachou et al. “EVCA Classifier: A MCMC-Based Classifier for Analyzing High-Dimensional Big Data”. In: *Information (Switzerland)* 14 (8 2023). Cited by: 4; All Open Access, Gold Open Access. DOI: [10.3390/info14080451](https://doi.org/10.3390/info14080451). URL: <https://www.scopus.com/inward/record.uri?eid=2-s2.0-85168770721&doi=10.3390%2finfo14080451&partnerID=40&md5=2ccdab5160f778cde0be452745e2d4df>.
- [89] James Walker. *16 Most Useful Container Orchestration Tools in 2024*. Dec. 2024. URL: <https://spacelift.io/blog/container-orchestration-tools>.
- [90] *What is Podman?* June 2025. URL: <https://docs.podman.io/en/latest/>.
- [91] HPC WIKI. “Slurm HPC Scheduler”. In: (June 2025). URL: <https://hpc.wiki.utwente.nl/slurm:start>.
- [92] wikipedia. *Deucalion (supercomputer)*. Dec. 2024. URL: [https://en.wikipedia.org/wiki/Deucalion_\(supercomputer\)](https://en.wikipedia.org/wiki/Deucalion_(supercomputer)).
- [93] Lennart Wissel. “LISA Pathfinder: Understanding DWS noise performance for the LISA mission”. In: *Journal of Physics: Conference Series*. Vol. 840. Cited by: 4; All Open Access, Gold Open Access, Green Open Access. 2017. DOI: [10.1088/1742-6596/840/1/012044](https://doi.org/10.1088/1742-6596/840/1/012044). URL: <https://www.scopus.com/inward/record.uri?eid=2-s2.0-85021231573&doi=10.1088%2f1742-6596%2f840%2f1%2f012044&partnerID=40&md5=d1218ab93bb3d597e26e39667c19b832>.
- [94] Andy B. Yoo, Morris A. Jette, and Mark Grondona. “SLURM: Simple Linux Utility for Resource Management”. In: 2003, pp. 44–60. DOI: [10.1007/10968987_3](https://doi.org/10.1007/10968987_3).

Appendix A

APC Globalfit Architecture description

Main Components

1. Modules

- **Location:** `modules/`
- **Key Classes:** `ModuleBase`, `StubModule`, `ModuleRegistry`
- **Role:**
 - Represent the core computational units (e.g., MCMC samplers).
 - Each module processes data, maintains state, and communicates via messages.
 - `ModuleBase` is the abstract base; `StubModule` is a concrete example.
- **Communication:**
 - Receive configuration and iteration commands.
 - Publish results and status updates.
 - Communicate via an event bus (see below).

2. Engine

- **Location:** `engine/engine.py`, `mock_engine.py`
- **Key Classes:** `Engine`
- **Role:**
 - Orchestrates the workflow by managing modules and their execution.
 - Handles rule evaluation, collects results, and manages module states.
- **Communication:**
 - Subscribes to module status and data updates.
 - Sends commands to modules (e.g., terminate, configure, iterate).
 - Uses the event bus for all messaging.

3. Preprocessors

- **Location:** `modules/stub/preprocessor.py`
- **Key Classes:** `StubPreprocessor`
- **Role:**
 - Prepare input data and initial module configurations.
 - Can use checkpoints to resume or adjust processing.
- **Communication:**
 - Writes configuration files.
 - Publishes group configuration messages to the engine.

4. Event Bus

- **Location:** `bus.py`, `buses/`
- **Key Classes:** `EventBus`, `PullEventBus`, `NatsEventBus`, `NatsPullEventBus`
- **Role:**
 - Abstracts message passing between components.
 - Implements publish/subscribe using NATS as backend.
- **Communication:**
 - All inter-component communication (modules, engine, CLI, etc.) happens via the bus.

5. Pipelinerunner

- **Location:** `pipelinerunner/`
- **Key Classes:** `WorkflowSpec`, `WorkflowSubmitter`
- **Role:**
 - Handles workflow submission and orchestration.
 - Bridges between workflow definitions and the event bus.
- **Communication:**
 - Publishes workflow specs to the bus.
 - Listens for workflow results and callbacks.

6. Monitoring

- **Location:** `monitoring/`
- **Key Files:** `service.py`, `api.py`, `ui.py`
- **Role:**
 - Provides APIs and UI for monitoring the state and results of workflows and modules.
 - Fetches checkpoints and status from storage (e.g., S3).
- **Communication:**
 - Reads from storage (S3) and exposes HTTP endpoints.
 - Does not directly interact with the event bus, but visualizes the results of bus-mediated computation.

7. CLI

- **Location:** `cli.py`

- **Role:**
 - Entry point for running and managing workflows, modules, and preprocessing.
 - Parses arguments and invokes the appropriate commands.
- **Communication:**
 - Uses the event bus to communicate with modules and engine.
 - Reads/writes configuration and checkpoint files.

8. Catalogs

- **Location:** `catalogs/`
- **Role:**
 - Manage source catalogs (e.g., memory, sqlite).
 - Provide source parameters to modules.
- **Communication:**
 - Used by modules and engine for source information.

9. Messaging

- **Location:** `msg/`
- **Key Files:** `base.py`, `control.py`, `data.py`, `subjects.py`
- **Role:**
 - Defines message types and subjects for communication.
 - Encodes/decodes messages for the event bus.
- **Communication:**
 - All messages exchanged via the event bus are defined here.

Communication Flow

1. **CLI → Preprocessor**
CLI invokes the preprocessor to generate group/module configurations and preprocessed data.
2. **Preprocessor → Engine**
Preprocessor publishes a `ModuleGroupConfiguration` message to the engine via the event bus.
3. **Engine → Modules**
Engine listens for group configuration messages.
Engine sends `ConfigureModule` and `IterateModule` messages to modules via the bus.
4. **Modules → Engine**
Modules publish `ModuleState` (results/checkpoints) and `ExecutionStateUpdate` (status) messages to the engine via the bus.
5. **Engine → Monitoring**
Engine uploads checkpoints to S3 (or similar storage).
Monitoring service reads checkpoints and status from storage for visualization.
6. **Pipelinrunner**
Submits workflow specs to the bus.
Handles workflow callbacks and results, publishing them as needed.

Summary Table

Component	Main Role	Communicates With	How (Direction)
CLI	User interface, orchestration	Preprocessor, Engine	Calls, bus messages
Preprocessor	Data/config preparation	Engine	Publishes config
Engine	Orchestration, rule evaluation	Modules, Monitoring	Publishes/subscribes
Modules	Computation, state management	Engine	Publishes/subscribes
Event Bus	Messaging backbone	All	Pub/Sub
Pipelinrunner	Workflow submission	Engine, Bus	Publishes/subscribes
Monitoring	Visualization, status	Storage, Engine	Reads, HTTP API
Catalogs	Source info	Modules, Engine	Function calls
Messaging	Message definitions	All	Imports

Example Communication Sequence

1. User runs CLI to preprocess data.
 2. Preprocessor writes configuration and publishes group config to the Engine.
 3. Engine creates module pool, sends configure/iterate commands to Modules.
 4. Modules run, publish results and status to Engine.
 5. Engine uploads checkpoints to S3.
 6. Monitoring reads checkpoints and displays results.
 7. Pipelinerunner can submit workflows and handle results via the bus.
-

In summary:

All major components interact via the event bus, using well-defined message types and subjects. The engine orchestrates the workflow, modules perform computation, the preprocessor prepares data, and the monitoring system visualizes results. The CLI and pipelinerunner provide user-facing entry points. Storage (e.g., S3) is used for checkpointing and monitoring.

Appendix B

Code Listing: First script

Listing B.1: First Run of the APC

```
1  #!/bin/bash
2
3  # Set up environment
4  source /home/babo/tese/framework/.venv/bin/activate
5
6  # Check if /tmp/globalfit_demo directory exists, if not delete it
7  if [ -d "/home/babo/tese/framework/tmp/globalfit_demo" ]; then
8      echo "Directory /tmp/globalfit_demo exists. Deleting it."
9      rm -rf /home/babo/tese/framework/tmp/globalfit_demo
10 else
11     echo "Directory /home/babo/tese/framework/tmp/globalfit_demo does not exist.
        Creating it."
12 fi
13
14
15 mkdir -p /home/babo/tese/framework/tmp/globalfit_demo
16 cd /home/babo/tese/framework/tmp/globalfit_demo
17
18 # Start NATS server
19 docker run -d --name nats -p 4222:4222 nats:2.10 --jetstream
20
21 # Simulate signal
22 lgfk-simulate-signal --n-gaussians 1 --n-samples 1 --snr 2 --output-path
        simulated_signal.h5
23
24 echo "-----"
25 echo "-----Starting iteration 0-----"
26 echo "-----"
27
28 # Step 1: Create the expected groups file
29 echo '[{"label":"stub"}]' > groups.json
```

```

30
31 # Step 2: Start the engine in the background
32 lgfk-cli \
33   --servers nats://localhost:4222 \
34   --pipeline-run-id "iter0" \
35   --log-level DEBUG \
36   start-engine \
37   --expected-groups groups.json \
38   --output-checkpoint checkpoint.json \
39   --output-iteration-decision iteration_decision.json \
40   --max-iterations 2 \
41   --current-iteration 0 &
42
43 # Step 3: Preprocess the signal
44 lgfk-cli \
45   --servers nats://localhost:4222 \
46   --pipeline-run-id "iter0" \
47   --log-level DEBUG \
48   preprocess \
49   --input-data simulated_signal.h5 \
50   --output-preprocessed-signal simulated_signal_preprocessed.h5 \
51   --output-group-configuration-path group_config_stub.json \
52   --group-label stub
53
54 # Step 4: Start the modules
55 lgfk-cli \
56   --servers nats://localhost:4222 \
57   --pipeline-run-id "iter0" \
58   --log-level DEBUG \
59   start-modules \
60   --input-data simulated_signal_preprocessed.h5 \
61   --input-group-configuration-path group_config_stub.json \
62   --current-iteration 0
63
64 # Step 5: Wait for the engine to complete
65 wait
66
67 echo "-----"
68 echo "-----Starting iteration 1-----"
69 echo "-----"
70
71 # Step 1: Create the expected groups file
72 echo '[{"label":"stub"}]' > groups.json
73
74 # Step 2: Start the engine in the background
75 lgfk-cli \
76   --servers nats://localhost:4222 \
77   --pipeline-run-id "iter0" \
78   --log-level DEBUG \

```

```
79  start-engine \  
80  --expected-groups groups.json \  
81  --output-checkpoint checkpoint.json \  
82  --output-iteration-decision iteration_decision.json \  
83  --max-iterations 20 \  
84  --current-iteration 1 &  
85  
86  # Step 3: Preprocess the signal  
87  lgfk-cli \  
88  --servers nats://localhost:4222 \  
89  --pipeline-run-id "iter0" \  
90  --log-level DEBUG \  
91  preprocess \  
92  --input-data simulated_signal.h5 \  
93  --output-preprocessed-signal simulated_signal_preprocessed.h5 \  
94  --output-group-configuration-path group_config_stub.json \  
95  --group-label stub  
96  
97  # Step 4: Start the modules  
98  lgfk-cli \  
99  --servers nats://localhost:4222 \  
100  --pipeline-run-id "iter0" \  
101  --log-level DEBUG \  
102  start-modules \  
103  --input-data simulated_signal_preprocessed.h5 \  
104  --input-group-configuration-path group_config_stub.json \  
105  --current-iteration 1  
106  
107  # Step 5: Wait for the engine to complete  
108  wait  
109  
110  docker stop nats  
111  docker rm nats
```

Appendix C

Code Listing: Slurm batch script

Listing C.1: Slurm batch script

```
1  #!/bin/bash
2  #SBATCH --partition=normal-arm
3  #SBATCH --job-name=globalfit_demo_simple
4  #SBATCH --account=f202500007hpcvlabuportoa
5  #SBATCH --time=00:20:00
6  #SBATCH --nodes=1
7  #SBATCH --ntasks=1
8  #SBATCH --cpus-per-task=48
9  #SBATCH --mem=16G
10 #SBATCH --output=globalfit_demo_simple_%j.out
11
12 source /share/env/module_select.sh
13 module load Python/3.12.3-GCCcore-13.3.0
14
15 source /projects/F202500007HPCVLABUPORTO/babo/framework/venv_arm/bin/activate
16
17 which python
18 python --version
19 # Set paths
20 NATS_BIN="/projects/F202500007HPCVLABUPORTO/babo/framework/nats-server-v2.11.3-
    linux-arm64/nats-server"
21 TMP_DIR_NATS="/projects/F202500007HPCVLABUPORTO/babo/framework/tmp/nats"
22 TMP_DIR_GLOBALFIT="/projects/F202500007HPCVLABUPORTO/babo/framework/tmp/
    globalfit_demo"
23
24 # Validate binary
25 if [ ! -f "$NATS_BIN" ]; then
26     echo "NATS binary not found at $NATS_BIN" >&2
27     exit 1
28 fi
29
```

```

30 if [ -d "$TMP_DIR_NATS" ]; then
31     echo "Temporary directory not found at $TMP_DIR_NATS" >&2
32     rm -rf "$TMP_DIR_NATS"
33 fi
34
35 mkdir -p "$TMP_DIR_NATS"
36
37 # Check if TMP_DIR_GLOBALFIT directory exists, if not delete it
38 if [ -d "$TMP_DIR_GLOBALFIT" ]; then
39     echo "Directory $TMP_DIR_GLOBALFIT exists. Deleting it."
40     rm -rf $TMP_DIR_GLOBALFIT
41 else
42     echo "Directory $TMP_DIR_GLOBALFIT does not exist. Creating it."
43 fi
44
45 mkdir -p $TMP_DIR_GLOBALFIT
46 cd $TMP_DIR_GLOBALFIT
47
48 # Start NATS server with JetStream
49 "$NATS_BIN" -js -sd "$TMP_DIR_NATS" &
50 NATS_PID=$!
51
52 trap "kill $NATS_PID" EXIT SIGINT SIGTERM
53
54 # Wait for server startup (5-10 seconds recommended)
55 sleep 1
56
57 # Simulate signal
58 lgfk-simulate-signal --n-gaussians 20 --n-samples 36 --snr 2 --output-path
    simulated_signal.h5
59 # lgfk-simulate-signal --output-path simulated_signal.h5
60
61 echo "-----"
62 echo "-----Starting iteration 0-----"
63 echo "-----"
64
65 # Step 1: Create the expected groups file
66 echo '[{"label":"stub"}]' > groups.json
67
68 # Step 2: Preprocess the signal
69 lgfk-cli \
70     --servers nats://localhost:4222 \
71     --pipeline-run-id "iter0" \
72     preprocess \
73     --input-data simulated_signal.h5 \
74     --output-preprocessed-signal simulated_signal_preprocessed.h5 \
75     --output-group-configuration-path group_config_stub.json \
76     --group-label stub
77

```

```

78 # Step 3: Start the modules
79 lgfk-cli \
80   --servers nats://localhost:4222 \
81   --pipeline-run-id "iter0" \
82   start-modules \
83   --input-data simulated_signal_preprocessed.h5 \
84   --input-group-configuration-path group_config_stub.json \
85   --current-iteration 0
86
87 # Step 4: Start the engine in the background
88 lgfk-cli \
89   --servers nats://localhost:4222 \
90   --pipeline-run-id "iter0" \
91   start-engine \
92   --expected-groups groups.json \
93   --output-checkpoint checkpoint.json \
94   --output-iteration-decision iteration_decision.json \
95   --max-iterations 2 \
96   --current-iteration 0
97
98 echo "-----"
99 echo "-----Starting iteration 1-----"
100 echo "-----"
101
102 # Step 1: Create the expected groups file
103 echo '[{"label":"stub"}]' > groups.json
104
105 # Step 2: Preprocess the signal
106 lgfk-cli \
107   --servers nats://localhost:4222 \
108   --pipeline-run-id "iter0" \
109   preprocess \
110   --input-data simulated_signal.h5 \
111   --output-preprocessed-signal simulated_signal_preprocessed.h5 \
112   --output-group-configuration-path group_config_stub.json \
113   --group-label stub
114
115 # Step 3: Start the modules
116 lgfk-cli \
117   --servers nats://localhost:4222 \
118   --pipeline-run-id "iter0" \
119   start-modules \
120   --input-data simulated_signal_preprocessed.h5 \
121   --input-group-configuration-path group_config_stub.json \
122   --current-iteration 1
123
124 # Step 4: Start the engine in the background
125 lgfk-cli \
126   --servers nats://localhost:4222 \

```

```
127 --pipeline-run-id "iter0" \  
128 start-engine \  
129 --expected-groups groups.json \  
130 --output-checkpoint checkpoint.json \  
131 --output-iteration-decision iteration_decision.json \  
132 --max-iterations 2000 \  
133 --current-iteration 1  
134  
135 kill $NATS_PID
```

Appendix D

Install, run and test the GlobalFit framework Locally

D.1 Local Deployment

Since this project is in the beginning stage, there is no documentation on running or using this tool. So the first step is to look at the actual code and determine what it does and how to use it.

After analysing the code, we wrote a documentation PDF to explain the flow of the tool and how to use each module separately. Then, we used the whole tool and created a simple workflow on the local computer. The PDF is in annex [A](#)

As we can see, using the CLI, we first need to create a signal, and then after it, we preprocess it, then run the engine, and only after we open the modules so that it can compute an MCMC.

D.1.1 How to install the framework

This code is written in Python, and it's a package. For the best reproducibility, the recommended way from ESA to install this tool is to use UV from Astral. This code also requires an event/message bus so that the modules can communicate with each other, as we can see in [4.1](#). The chosen event bus is NATS with the extension Jetstream. The code referring to the pipeline runner in [4.1](#) is deployed using an instance of Kubernetes using ARGO Workflows. In this case, Docker with Minikube was used for local deployment, and an example of Argo in the K8s cluster is sufficient to test the tool.

After installing and connecting the external tools outside of this tool, we can try to run it locally.

D.1.2 First Run of the APC

After multiple tries and errors, we created a Shell script that successfully runs a trivial case of a fit of one Gaussian in a local computing environment. The script is in the annex [B](#). This script runs successfully on the first run, stores a checkpoint, and then runs again, starting over the checkpoint.

And successfully splitting this job over the processes and then joining them in the end, utilizing all the CPU capabilities. As we can see from the following two pictures. (logs and CPU resources utilization)

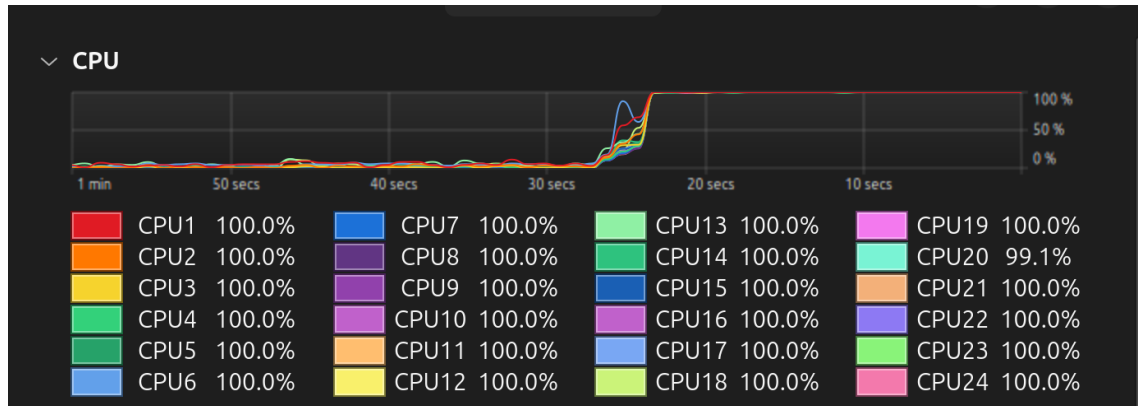


Figure D.1: CPU utilization during the first run.

```

babo@babo-MS-7C56:~/tese/framework$ ./run.sh
Directory /tmp/globalfit_demo exists. Deleting it.
96e2655fc88eabc084ffd31fc26f990d84064b3ba072b70209a2f8fc77035b04
-----
-----Starting iteration 0-----
-----
2025-06-16T10:48:54.524 [INFO] lisa.globalfit.framework.cli running stub preprocessor
2025-06-16T10:48:54.526 [INFO] lisa.globalfit.framework.preprocessing wrote group configuration to group_config_stub.json
2025-06-16T10:48:54.526 [WARNING] lisa.globalfit.framework.cli TODO: group-level subtraction of signals
2025-06-16T10:48:54.584 [INFO] lisa.globalfit.framework.mock_engine got configuration of 2 modules for group stub
2025-06-16T10:48:54.587 [INFO] lisa.globalfit.framework.mock_engine starting engine
2025-06-16T10:48:54.587 [INFO] lisa.globalfit.framework.mock_engine running iteration 0/1
2025-06-16T10:48:54.587 [INFO] lisa.globalfit.framework.mock_engine 0/2 modules iterated
2025-06-16T10:48:55.283 [INFO] lisa.globalfit.framework.cli starting 2 module subprocesses
2025-06-16T10:48:55.289 [INFO] lisa.globalfit.framework.modules.module env variable S3_BUCKET is not set, checkpoints will not be uploaded to S3
2025-06-16T10:48:55.292 [INFO] lisa.globalfit.framework.modules.module env variable S3_BUCKET is not set, checkpoints will not be uploaded to S3
2025-06-16T10:48:55.589 [INFO] lisa.globalfit.framework.mock_engine 0/2 modules iterated
2025-06-16T10:48:56.590 [INFO] lisa.globalfit.framework.mock_engine 0/2 modules iterated
2025-06-16T10:48:57.592 [INFO] lisa.globalfit.framework.mock_engine 0/2 modules iterated
2025-06-16T10:48:57.874 [INFO] lisa.globalfit.framework.modules.stub.module sci.gf.iter0.stub.16 iteration 0 best sample: [15.82920008 7.36567473] logprob: -5.2700317959529315
2025-06-16T10:48:57.909 [INFO] lisa.globalfit.framework.modules.stub.module sci.gf.iter0.stub.28 iteration 0 best sample: [27.79396004 8.81625198] logprob: -5.212383353646047
2025-06-16T10:48:58.593 [INFO] lisa.globalfit.framework.modules.pool wrote module pool checkpoint checkpoint.json
2025-06-16T10:48:58.593 [INFO] lisa.globalfit.framework.mock_engine all modules finished iteration
2025-06-16T10:48:58.595 [INFO] lisa.globalfit.framework.catalogs.sqlite wrote 2 source parameters to catalog.json
2025-06-16T10:48:58.597 [INFO] lisa.globalfit.framework.mock_engine wrote iteration decision to iteration_decision.json

```

Figure D.2: Logs of the first run.

As shown in Figure D.2, we have successfully run the code, achieved a result, and saved the result that can later be used for a second run. And in Figure D.1 we verify that all cores are being used, and we used a 100% CPU utilization. This means that the tool could take advantage of the whole processing unit, utilizing parallel execution, and then merge the results back into one thread.

D.1.3 Second Run of the APC

To increase the complexity and complicate the calculation to simulate a normal load circumstance in the future more accurately, it is possible to do so by complicating the input signal. In Annex B, we can look at line 22 and change it to:

```
1 lgfk-simulate-signal --n-gaussians 2000 --n-samples 3650 --snr 200 --output-path  
   simulated_signal.h5
```

This will significantly increase the time each block takes to run, and alone, it will make the whole scripted run take about 10x more time. This is sufficient to test the infrastructure and verify that it works correctly.

The next logical step is to run this framework in a real environment, an HPC. The HPC where we will test is the new Portuguese EuroHPC Deucalion, and we will dive into its architecture and features to better understand how we can deploy it.

Appendix E

Globalfit framework in Deucalion

E.1 Access and Install dependencies

After configuring an SSH key as described in the documentation [57], we will enter a login node running an x86 architecture. We must change the directory to the projects directory on the fast Infiniband filesystem to install and run projects using Slurm. Our approved project name from FCT is **F202500007HPCVLABUPORTO**, so that's where we will run our experiments.

Since this is an HPC, there are no tools to get or install packages, and all one has to do is either download a binary or compile the code to work in Deucalion. First, we will discuss which steps we must take, then go into depth on each.

1. Download the proprietary code of the GlobalFit framework
2. Install the dependencies and create an environment using Slurm.

Note: This is because we only have access to arm64 CPU hours. And because the login node runs on a different architecture, if we try to install it on the login node, it will not run on the Slurm jobs. This means we have to create a Slurm job to install and set up the environment before being able to run the framework.

3. Download the correct arm64 binaries for NATS and ensure it can run inside Slurm.

E.1.1 Installation of the Framework in Deucalion

The code is not open source and is in an ESA GitLab repository, which I can access. I set a unique token and downloaded version 0.4.0 using git via HTTPS. After having the code in Deucalion, we need to install it using Slurm to accommodate Slurm dependencies. The first step is to ask Slurm to allocate a job for us, and we will SSH into it, and then the second step is to use the shell to compile and install the dependencies. We can do the first step using the following command:

```
1 srslun -A f202500007hpcvlabuportoa -t 4:00:00 -p dev-arm -N1 --pty bash
```

Breaking down this command, we use **srun** to ask Slurm to allocate a compute node and execute in interactive and blocking mode for us, we specify our account with our project name **f202500007hpcvlabuportoa**, and set a time of 4 minutes. The partition we have access to is **arm**, and in this case, we need the smaller partition, so we go with **dev-arm**. At the end of the command, we use **-pty bash** to make it so we instantly SSH to the compute node and interact with it.

After running this on the projects directory, we have to wait until Slurm automatically directs our shell to inside the Slurm job. After a successful allocation, the shell should look like this:

```

diogobabo@ln04:/projects/F202500007HPCVLABUPORTO/babo

Greetings diogobabo

Welcome to the Portuguese EuroHPC supercomputer, *please* read documentation before use:
https://docs.macc.fccn.pt/

=====
cdFilesystem      Space  Quota  Limit Files  Quota  Limit
/home/diogobabo    2406M  20480M 25600M 83367   200k   250k
=====
Disk quotas for prj 60168 (pid 60168):
  Filesystem      used  quota  limit  grace  files  quota  limit  grace
/projects/F202500007HPCVLABUPORTO
                  43.76G 9.766T 14.65T   - 180579      0      0      -
=====
[diogobabo@ln04 ~]$ cd /projects/F202500007HPCVLABUPORTO/babo
[diogobabo@ln04 babo]$ pwd
/projects/F202500007HPCVLABUPORTO/babo
[diogobabo@ln04 babo]$ ls
framework poc-maindcc-in-cloud
[diogobabo@ln04 babo]$ srun -A f202500007hpcvlabuportoa -t 4:00:00 -p dev-arm -N1 --pty bash
srun: job 462475 queued and waiting for resources
srun: job 462475 has been allocated resources
[diogobabo@cna0002 babo]$

```

Figure E.1: Accessing a compute node

After achieving the same as in Figure E.1, we can execute the commands we would typically use to compile and install the framework. Since we are in an HPC, we do not have access to the recommended way to install, that is, UV from Astral, as referred to before. So, the workaround solution is to create a simple Python environment with native Python, because this project is a package with a "**pyproject.toml**", we can use it without any problem. Also, as described in the documentation, we first need to source Python and native packages so that Python works inside Slurm. The commands used are the following:

```

1 source /share/env/module_select.sh
2 module load Python/3.12.3-GCCcore-13.3.0
3 python3 -m venv venv_arm
4 source venv_arm/bin/activate
5 pip install --upgrade pip setuptools wheel build
6 pip install .

```

After following the documentation, we chose a Python version that suits us. These are the first two lines. After creating a module, we activate it, install the build dependencies, and then install the module.

E.1.2 Download the binaries

For this testing purpose, we will only need a NATS binary that works with ARM, and we can find it in "<https://docs.nats.io/running-a-nats-service/introduction/installation>" where we will go to the release build and use curl to download it onto the machine. In this case, we used "**nats-server-v2.11.3-linux-arm64**" and that's the directory inside the project where we have unzipped the binary.

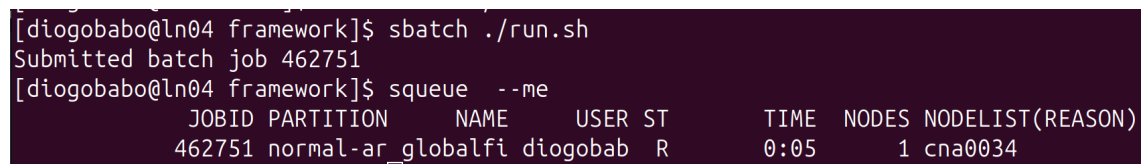
E.2 Run the framework on Deucalion

Having all dependencies working and all the binaries in place, based on the annex B, we can code and write a Slurm batch script that will automatically allocate the resources, run the framework, and output the result. After trial and error, we arrived at the code in the annex C. We automatically deal with all the Slurm requirements and ask to schedule a job at the beginning of the script. Here, we also create a temporary directory storing the logs and results. We source the binaries and trap the code to kill the NATS instance at the end of the job. And we can access the logs in real-time because a file is created where the logs are dumped, called "**globalfit_demo_simple.out**". Here we can trail it and have a live view of what's happening.

To run this code, we have to have the annex C inside the directory where the binaries and framework are, and then on the login node, ask to queue a Slurm job using batch. Using the following way:

```
1 sbatch ./run.sh
```

This will start the job as soon as the requested resources are available and begin writing to the file system. This code runs successfully and executes as intended, as we can see from the following two figures E.2 and E.3:



```
[diogobabo@ln04 framework]$ sbatch ./run.sh
Submitted batch job 462751
[diogobabo@ln04 framework]$ squeue --me
```

JOBID	PARTITION	NAME	USER	ST	TIME	NODES	NODELIST(REASON)
462751	normal-ar	globalfi	diogobab	R	0:05	1	cna0034

Figure E.2: Submitting a Slurm batch script

```

2025-06-16T16:13:18.047 [INFO] lisa.globalfit.framework.modules.stub.module sci.gf.iter0.stub.2 iteration 0 best sample: [2.27438379 5.28311467] logprob: -9.392643992201739
2025-06-16T16:13:18.095 [INFO] lisa.globalfit.framework.modules.stub.module sci.gf.iter0.stub.16 iteration 0 best sample: [15.95493661 5.25673058] logprob: -9.354613000519937
2025-06-16T16:13:18.208 [INFO] lisa.globalfit.framework.modules.stub.module sci.gf.iter0.stub.27 iteration 0 best sample: [27.40300401 5.21367349] logprob: -9.34247049849632
2025-06-16T16:13:18.357 [INFO] lisa.globalfit.framework.modules.stub.module sci.gf.iter0.stub.3 iteration 0 best sample: [3.40240051 8.0044432 ] logprob: -9.366064166276692
2025-06-16T16:13:18.402 [INFO] lisa.globalfit.framework.modules.stub.module sci.gf.iter0.stub.23 iteration 0 best sample: [22.73916694 7.02323231] logprob: -9.308342479223269
2025-06-16T16:13:18.414 [INFO] lisa.globalfit.framework.modules.stub.module sci.gf.iter0.stub.30 iteration 0 best sample: [29.79342554 3.0412774 ] logprob: -9.389276220112087
2025-06-16T16:13:18.421 [INFO] lisa.globalfit.framework.modules.stub.module sci.gf.iter0.stub.20 iteration 0 best sample: [19.98946676 2.20361431] logprob: -9.397880768044104
2025-06-16T16:13:18.440 [INFO] lisa.globalfit.framework.modules.stub.module sci.gf.iter0.stub.35 iteration 0 best sample: [35.1100644 2.74809042] logprob: -9.413559064420717
2025-06-16T16:13:18.482 [INFO] lisa.globalfit.framework.modules.stub.module sci.gf.iter0.stub.33 iteration 0 best sample: [33.34675403 9.71381684] logprob: -9.333649259443842
2025-06-16T16:13:18.590 [INFO] lisa.globalfit.framework.modules.stub.module sci.gf.iter0.stub.13 iteration 0 best sample: [13.37873649 7.71262546] logprob: -9.322311412830494
2025-06-16T16:13:18.630 [INFO] lisa.globalfit.framework.modules.stub.module sci.gf.iter0.stub.31 iteration 0 best sample: [30.94039599 9.75390804] logprob: -9.315931891851237
2025-06-16T16:13:18.646 [INFO] lisa.globalfit.framework.modules.stub.module sci.gf.iter0.stub.5 iteration 0 best sample: [4.61455646 2.42228727] logprob: -9.407436289752496
2025-06-16T16:13:18.678 [INFO] lisa.globalfit.framework.modules.stub.module sci.gf.iter0.stub.8 iteration 0 best sample: [8.16368059 2.73061166] logprob: -9.398467565821681
2025-06-16T16:13:18.683 [INFO] lisa.globalfit.framework.modules.stub.module sci.gf.iter0.stub.28 iteration 0 best sample: [28.29868839 1.41373846] logprob: -9.40967013290711
2025-06-16T16:13:18.795 [INFO] lisa.globalfit.framework.modules.stub.module sci.gf.iter0.stub.25 iteration 0 best sample: [25.08630648 9.03061197] logprob: -9.290357027163186
2025-06-16T16:13:25.638 [INFO] lisa.globalfit.framework.mock_engine got configuration of 15 modules for group stub
2025-06-16T16:13:25.659 [INFO] lisa.globalfit.framework.mock_engine starting engine
2025-06-16T16:13:25.659 [INFO] lisa.globalfit.framework.mock_engine running iteration 0/1
2025-06-16T16:13:25.660 [INFO] lisa.globalfit.framework.mock_engine 0/15 modules iterated
2025-06-16T16:13:26.661 [INFO] lisa.globalfit.framework.mock_engine 11/15 modules iterated
2025-06-16T16:13:27.682 [INFO] lisa.globalfit.framework.modules.pool wrote module pool checkpoint checkpoint.json
2025-06-16T16:13:27.682 [INFO] lisa.globalfit.framework.mock_engine all modules finished iteration
2025-06-16T16:13:27.696 [INFO] lisa.globalfit.framework.catalogs.sqlite wrote 15 source parameters to catalog.json
2025-06-16T16:13:27.706 [INFO] lisa.globalfit.framework.mock_engine wrote iteration decision to iteration_decision.json

```

Figure E.3: Logs of the execution of the Slurm script. Confirming the successful run.