EUPEX
European Pilot for Exascale

D5.1 – Software Specification

Document Properties

Contract Number	101033975
Contractual Deadline	M17 (May 31, 2023)
Dissemination Level	Public
Nature	Report
Editors	S. Pickartz (ParTec), T. Moschny (ParTec)
Authors	P. Couvée, P. Lemarinier (Atos), A. Cedeyn (CEA), G. Agosta, D. Cattaneo, W. Fornaciari, V. Zaccaria (CINI-POLIMI), A. Bartolini (CINI-UNIBO), M. Torquati (CINI-UNIPI), M. Aldinucci, I. Colonnelli, A. R. Martinelli (CINI-UNITO), F. Guimarães, B. Mohr (FZJ), B. Goglin (Inria), C. Clauss, S. Krempel, T. Moschny, S. Pickartz, M. Rauh (ParTec), N. Bartelheimer, S. Neuwirth (UniFRA), O. Vysocký (VSB-TUO),
Reviewers	F. Magugliani (E4), J.-R. Bacou (Atos)
Date	May 31, 2023
Keywords	EUPEX, HPC, Exascale, Software
Status	Final
Release	1.0

EuroHPC
Joint Undertaking

This project has received funding from the European High-Performance Computing Joint Undertaking (JU) under grant agreement No 101033975. The JU receives support from the European Union's Horizon 2020 research and innovation programme and France, Germany, Italy, Greece, United Kingdom, Czech Republic, Croatia.

History of Changes

Release	Date	Author, Organisation	Description of Changes
1.0	30.05.2023	T. Moschny (ParTec)	Final version

Table of Contents

List of Figures

1 Introduction

The modular design of the European Pilot for eXascale (EUPEX) platform has a strong impact on the architecture of the software ecosystem. The Modular Supercomputer Architecture (MSA) adds an additional level in the hierarchy of traditional supercomputing systems by introducing the concept of tightly-coupled *modules*. Each module, in turn, is a parallel computer[1] with hardware characteristics satisfying the requirements of a specific type of application or parts thereof. The tight coupling is ensured by means of a *federated* high-speed network joining potentially different, module-specific fabrics.

This architectural approach comes with advantages compared to traditional designs and challenges regarding the whole software stack ranging from the management layer up to the application level. The notion of modules enables MSA systems to evolve over time, i. e., instead of disassembling a system after its lifetime, modules can be added and/or removed. However, this requires support of the management software providing the ability to insert or remove modules with minimal effort and impact on the system's operation, i. e., large system downtimes have to be avoided. The execution environment has to be capable of providing an efficient access to all system resources to the applications. On the one hand, this requires the ability to execute parallel workloads transparently across multiple modules. On the other hand, it should also allow applications to leverage topological information on the MSA hierarchy for further optimisation. This requires close interaction with the runtime environment providing Application Programming Interfaces (APIs) for accessing this information.

Power management, energy efficiency, and performance optimisation are crucial to achieve Exascale performance while meeting a reasonable power budget. Therefore, tools for monitoring

[1]This also includes quantum modules.

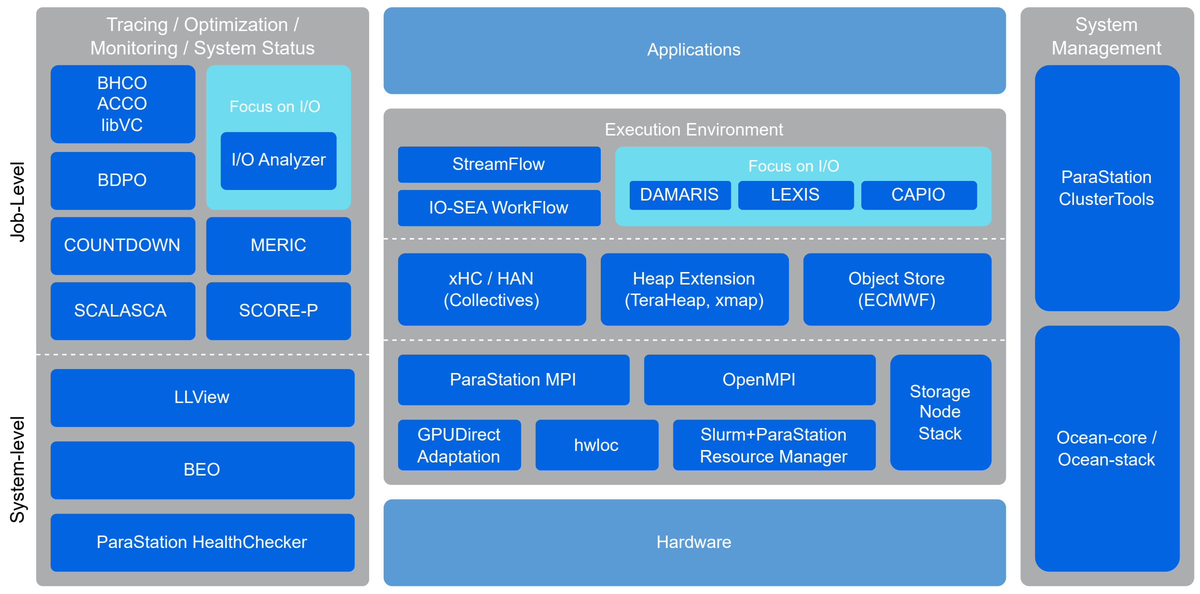


FIGURE 1: THE EUPEX SOFTWARE ECOSYSTEM COVERING ALL ASPECTS RANGING FROM SYSTEM MANAGEMENT, VIA THE EXECUTION ENVIRONMENT, THROUGH TO TRACING/MONITORING/PROFILING AND INPUT / OUTPUT (I/O).

and profiling have to be adapted to the design of the EUPEX platform covering all levels ranging from the system level, via the node level through to the application level. Finally, the I/O system is also affected by the modular system design. Instead of a single, central storage, MSA system exhibit a multi-level storage hierarchy with different characteristics. The system software faces the challenge of providing fast and scalable access to data for heterogeneous workflows and applications while minimising data transfers across the different storage tiers.

This deliverable introduces the software ecosystem of the EUPEX platform (cf. Fig. 1) and details how the above-mentioned challenges are met. Chapter 2 presents components of the management software stack. Chapter 3 introduces the fully MSA-aware execution environment enabling a flexible and efficient utilisation of the available resources within the EUPEX platform. Chapter 4 deals with tools for performance and energy efficiency that allow for monitoring and profiling the applications' power/energy consumption and to determine performance bottlenecks. And finally, Chapter 5 presents the elements of the I/O system enabling fast and scalable access to data within the hierarchical storage system.

2 Management Software Stack

The role of T5.1 is to provide a management software stack for the EUPEX Pilot, which meets the requirements of the MSA [1, 2] and hence will also enable the management of future MSA-based supercomputers. While traditional management software stacks used to deal with individual cluster for login, services and compute nodes, the MSA-aware management stack will need to embrace the whole picture to efficiently and coherently operate a set of strongly linked modules. This management software stack will not be build from scratch, but rely on the integration of different software pieces:

- CEA's management stack Ocean
- ParTec's cluster management suite ParaStation Modulo
- Elements to ensure a Trusted Execution Environment (TEE) provided by Foundation for Research and Technology – Hellas (FORTH)
- LLview as a "portal" for making lower level information from the computing nodes available to the users provided by Forschungszentrum Jülich GmbH

The main effort in T5.1 will be about the integration of these components into a coherent and efficient cluster management stack for the EUPEX Pilot. We will first provide some details on the requirements for such a software stack. This is followed by a description of the different components of the stack, along with a planning of the different elements to be done as well as their integration into a single management stack for the EUPEX Pilot.

2.1 Control Plane

2.1.1 Ocean

Ocean [3] is an open source suite for High Performance Computing (HPC) cluster administration. It is developed by Commissariat à l'énergie atomique et aux énergies alternatives (CEA) and supports (1) the deployment phase and (2) the day-to-day management of large HPC systems. In its current version, Ocean is already used as administration stack on all the different HPC clusters on CEA's premises, including the Tier-0 system Irene [4]. Ocean will be developed further within the EUPEX project, as to serve also the EUPEX Pilot and its particular needs.

Roughly speaking, Ocean consists of two parts: (1) Ocean-core and (2) Ocean-stack. *Ocean-core* is an RPM-based Linux distribution tailored to HPC cluster administration. This distribution is built upon a standard Linux distribution and gathers a large range of open source tools, allowing for an almost automated management of heterogeneous HPC clusters. Today, Ocean-core is based on CentOS and will move to Almalinux for future versions.

In addition, *Ocean-stack* describes design and implementation guidelines for such HPC clusters, along with best practices and standard operating procedures. This applies to main services management (such as deployment, content delivery, configuration management, hardware management, etc.) in a robust way. Ocean-stack is the handbook to build and operate HPC clusters at scale.

The suggested reference deployment scheme for the EUPEX Pilot is shown in Figure 2. It consists of a set of services provided for the entire cluster. These services are hosted on the machines called TOP1, TOP2, TOP3 and WORKER1, WORKER2, and WORKER3 constituting the *core administration service*.

In addition, each cluster type (login, compute, storage) is connected to the core administration service through two or more administrative nodes, so-called *islets* (ISLET10, ISLET11 for one cluster and ISLET20, ISLET21 for another one). Each islet will have for some services its local instances (e. g., for the boot and installation services). With this kind of architecture, each new cluster can be added with its own resources connected to the *core administration service* providing a cached level with a centralised administration. This approach does scale well and is currently in production on the CEA sites with 16 islets and 5000 nodes.

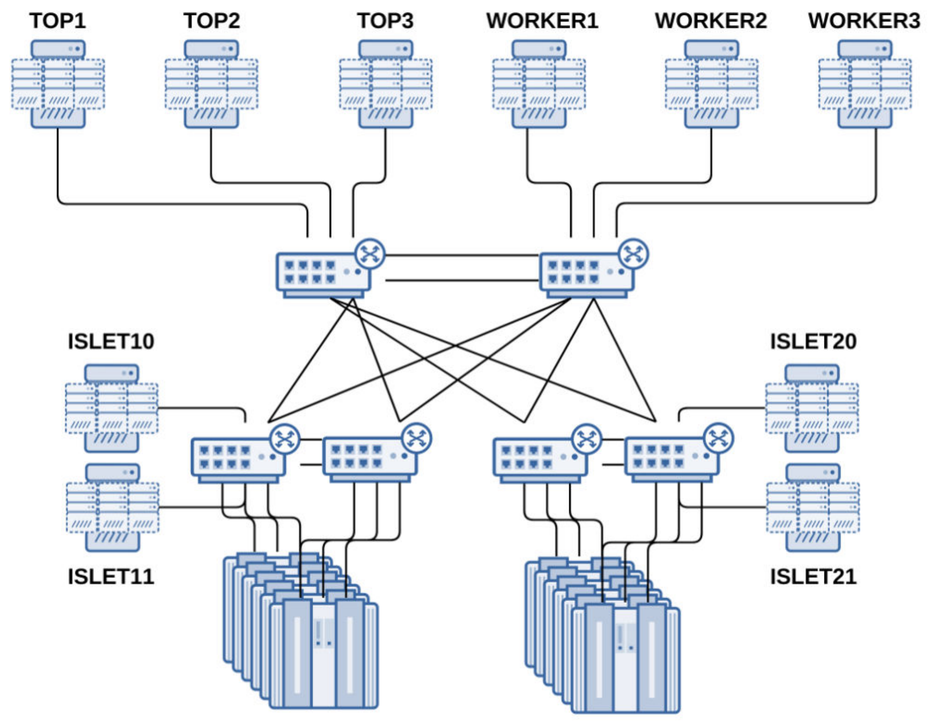


FIGURE 2: THE REFERENCE DEPLOYMENT SCHEME FOR THE EUPEX PILOT BASED ON THE OCEAN ARCHITECTURE.

Developments/Extensions planned within EUPEX

In the Ocean project, we plan to provide some evolutions regarding EUPEX:

- Support the EUPEX Central Processing Unit (CPU) architecture in Ocean-Core
- Describe the EUPEX CPU, Graphics Processing Unit (GPU), and interconnect installation and integration in Ocean-stack
- Create a collaborative platform to allow EUPEX community to contribute to the project

Timeline for development and for the integration

M24 › Updated Ocean version

M30 › Collaborative platform and tools created and available to the EUPEX community

› Ocean-core pre-release supporting aarch64

M36 › Final Ocean version with EUPEX dependencies

2.1.2 ParaStation Modulo

ParaStation Modulo is the MSA-enabling supercomputing software suite developed by ParTec. It is extensively used in production environments such as the Jülich Research on Exascale Cluster Architectures (JURECA) and Jülich Wizard for European Leadership Science (JUWELS) Cluster-Booster systems run by Jülich Supercomputing Centre (JSC) and the modular MeluXina system in Luxembourg.

This software suite provides a holistic system software stack for supercomputing, covering everything from system provisioning and management, through automated fault detection and problem tracking, to the provisioning of an execution environment and runtime system for large-scale, modular workloads. ParaStation Modulo is composed of four main pillars (cf. Fig. 3):

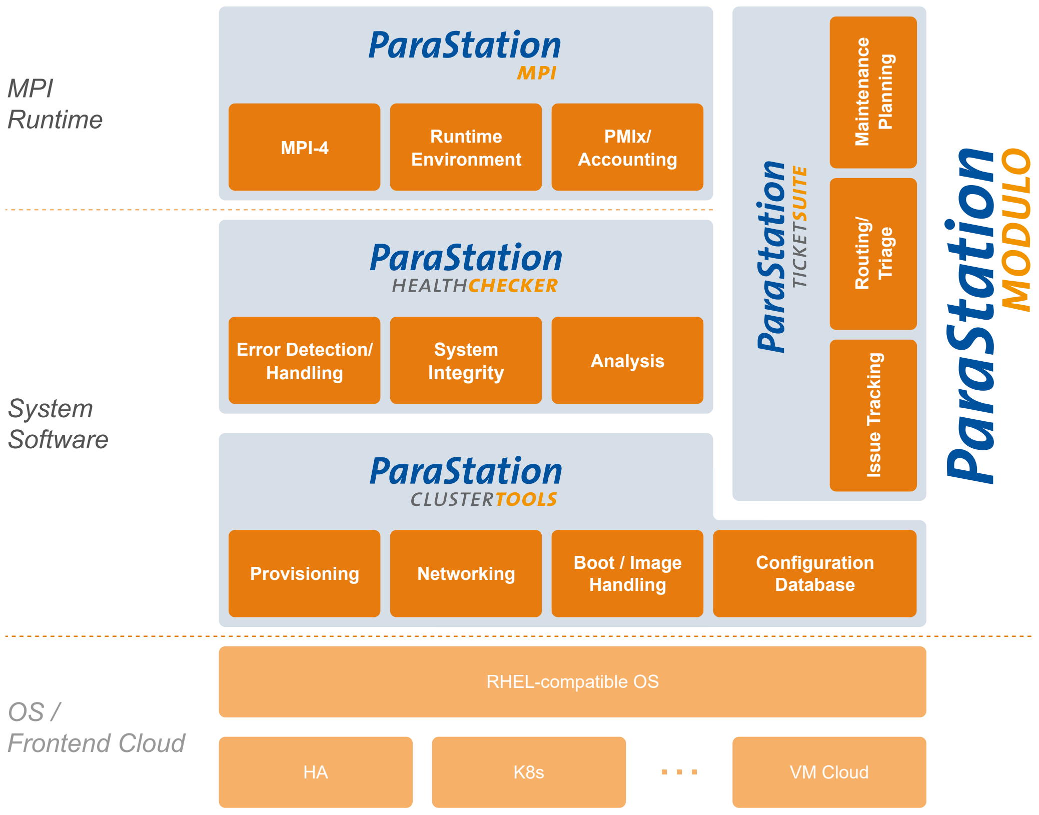


FIGURE 3: THE FOUR PILLARS OF THE PARASTATION MODULO SOFTWARE SUITE: PARASTATION CLUSER-TOOLS, PARASTATION HEALTHCHECKER, PARASTATION TICKETSUITE, AND PARASTATION MPI.

ParaStation ClusterTools This is a toolbox enabling modular cluster management, provisioning, and administration. Its image-based approach for node provisioning supports both diskless and diskfull installations while still supporting individual, per-node customisations.

ParaStation HealthChecker This is a framework enabling automated error detection and integrity checking on the global system level and on the node level. It comprises a comprehensive set of fast and non-destructive tests minimising the likelihood for node failures during job execution while maximising the system utilisation. Part of this framework is the so-called *checkbot* that regularly tries to bring back offlined nodes by eliminating transient errors.

ParaStation TicketSuite This is a powerful tool for issue tracking, analysis, maintenance planning, etc. It integrates with the ParaStation ClusterTools and the ParaStation Healthchecker. This way, it enables seamless support actions for on-site and off-site teams and constitutes the single point of contact for all parties involved in the utilisation and administration of a supercomputing system.

ParaStation MPI Is composed of a process manager and an Message-Passing Interface (MPI) runtime. Both components are tightly integrated, enabling efficient resource utilisation on HPC systems by large-scale, parallel applications. Its rich set of features especially targeting MSA architectures allows for the execution of modular workloads and workflows. By adhering to the Process Management Interface for Exascale (PMIx) standard, a scalable process start-up is ensured even for third-party runtimes and communication middlewares.

Developments/Extensions planned within EUPEX

The EUPEX management stack will be the result of the integration of ParaStation Modulo with Ocean. Therefore, the components of ParaStation Modulo have to be adapted to support the Ocean-core Linux distribution. This can already be achieved on the EUPEX Alfa Pilot platform (EAP) to a large extent. However, this integration has to be verified as soon as a first beta system is available to ensure the smooth interplay of both management stacks on the final pilot system.

The ParaStation ClusterTools have to be developed further to support the EUPEX hardware platform, e. g., access to the Baseboard Management Controller (BMC) is likely to differ from the interfaces commonly used for traditional x86-based systems. Additionally, the ParaStation HealthChecker will play an important role during installation and operation of the Pilot system. In this context, its set of tests will be extended and adapted to the platform, ensuring stable system operation.

Timeline for development and for the integration

M24	› Initial support for the Ocean-core Linux distribution
M30	› Collaborative platform and tools created and available to the EUPEX community
	› ParaStation Modulo pre-release supporting the EUPEX hardware platform
M42	› Final integration of ParaStation Modulo with Ocean meeting EUPEX dependencies

2.2 Trusted Execution and Monitoring

2.2.1 TEE-Enabled Storage Device Access

The lack of trust in third-party or shared resources within an HPC environment poses a significant barrier to distribute computations, especially when sensitive data is involved. As the importance of data increases for modern applications and workflows, protecting data within an HPC system from other applications and users becomes an important challenge. Even though current privacy-enhancing technologies, such as homomorphic encryption, can increase security, they are still very slow and can have very high storage requirements, especially for rich computations [5, 6]. Hardware-based TEEs are an attractive alternative that can provide trust guarantees for code execution in HPC computing environments and protect even against highly privileged adversaries [7].

TEEs derive from decades of similar efforts; however, it is only the recent years that vendors have started to provide hardware TEE support within commodity processors. The most popular cases include Advanced RISC Machines (ARM) Trust Zone [8], AMD SEV [9], and Intel SGX [10], each of which comes not only with different architectural designs and execution models, but with different software stacks and Software Development Kits (SDKs). As a matter of fact, SGX uses the SDK provided by Intel. ARM Trust Zone is typically available to only device manufacturers and operating systems; OP-TEE [11] is an open-source TEE that implements the ARM Trust Zone technology. AMD SEV can only be configured to protect whole Virtual Machine (VM) instances through transparent memory encryption.

Besides the low-level SDKs provided by vendors, there are also several third-party frameworks for developing TEE-enabled applications. The majority of these frameworks (e. g., SCONE,[1] Gramine,[2] and Occlum[3]) primarily aim to ease the development process by offering ready-to-use containers, unikernels, libraries and runtimes that allow existing applications to run (almost) unmodified. Other approaches, such as Google's Asylo and Microsoft's Open Enclave, focus on portability, by offering an universal interface for cross-platform TEE offloading. The latter approaches support cross-platform implementations, enabling the same implementation to run over different hardware TEEs. For example, Open Enclave provides support for Intel SGX and OP-TEE OS on ARM Trust Zone.

Developments/Extensions planned within EUPEX

Our approach will be based on Open Enclave SDK[4] and focus on designing a fully-functional key-value storage system that will be able to protect data from unwanted leaks. Key-value stores are designed to provide fast access to data using a simple key-value lookup system and can be useful for HPC applications that need to store and retrieve (i) large amounts of data quickly, such as data for simulations or real-time analytics systems and/or (ii) metadata associated with simulations or experiments, such as parameters, settings, or configurations. By implementing the

[1]https://scontain.com/
[2]https://gramineproject.io/
[3]https://occlum.io/
[4]https://openenclave.io/sdk/

functionality of a key-value persistent store within TEE, our design will further provide protection against security violations or attacks to compromise the stored data and the query operations.

Design Objectives. Even though TEEs aim to provide strong data protection during processing, it is also equally important to guarantee that data remains well protected through their whole life cycle, including when transiting over the network and when stored in main memory and/or storage devices. One of the most effective ways to protect data is by using encryption, which should be enforced end-to-end to the aforementioned phases (namely, transit phase, in-use phase, and rest phase). We briefly describe each of them below:

- **Data in transit:** The data is transferred over the network, in a client-server fashion (i. e., the client reads or writes key-value pairs on a remote server). In our case, the network is considered untrusted, hence the connection should be encrypted. This can be performed at different levels of the network stack, e. g., on application level by using SSL/TLS sockets, or at the network level, by establishing IPsec or VPN channels between the nodes. Unlike the traditional client-server architecture, in our case the operating system is also considered untrusted, hence, the encrypted connection should terminate within the TEE. By doing so, the data will only be decrypted within the protected space of the TEE, without being risked to be leaked in the network or the operating system.
- **Data in-use:** The data are protected during computation, using the hardware-based TEE (e. g., Intel SGX, ARM Trust Zone, etc.). TEEs provide an isolated space that is not accessible by other applications, the operating system, or even other hardware modules.
- **Data at rest:** The storage devices typically reside outside the trusted domain, hence the data should always be stored in an encrypted form. The encryption can be performed transparently, at the block-level (e. g., similar to dm-crypt), or at the application layer (e. g., similar to SQL TDE, etc.). In any case, it is necessary to redesign the data structures to extend the trust to the untrusted storage medium, in order to achieve end-to-end security properties between the TEE and the storage mediums.

Timeline for development and for the integration

M18 – Analysis of different design choices and how these choices could affect the security and performance of a TEE-based key-value storage system.

M36 – Provide the final design of a confidential key-value storage system, built on hardware-based trusted execution environment.

2.2.2 LLview

LLview[5] is a software tool that provides comprehensive monitoring and reporting of clusters that are controlled by a resource manager and a scheduler system. It includes an interactive role-based web portal that displays the jobs, their allocation to resources and also performance metrics obtained from other monitoring sources. The job reporting interface of the LLview web portal, shown in Fig. 4, displays a list of the currently running jobs and their data and metrics, updated every 1-3 minutes. LLview ensures that only previously existing data is collected from the

[5]https://apps.fz-juelich.de/jsc/llview/docu/

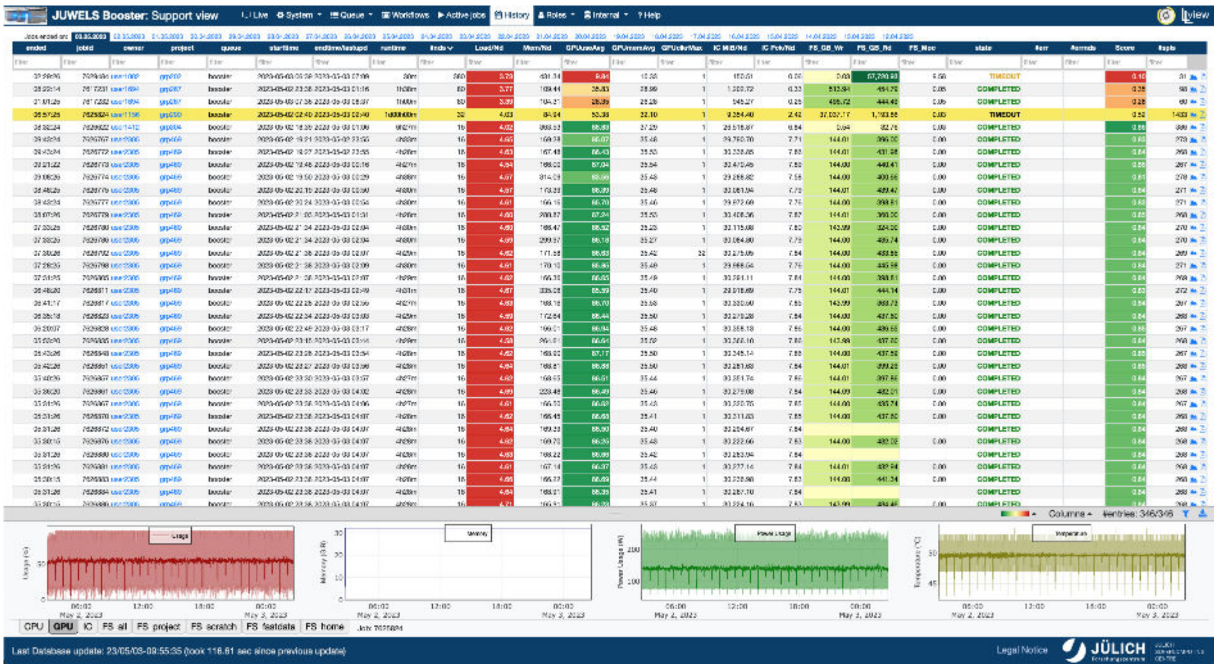


FIGURE 4: LLVIEW WEB PORTAL DISPLAYING JOBS THAT RAN ON THE MONITORED SYSTEM. WHEN A JOB IS SELECTED, SOME AVERAGED GRAPHS ARE SHOWN AT THE BOTTOM. DETAILED REPORTS IN HTML AND PDF CAN BE ACCESSED FROM THE TWO RIGHTMOST COLUMNS.

system, such as the resource manager and existing daemons. Further sources may be installed on the compute nodes with a minimal overhead to obtain metrics in the range of minutes.

The LLview Job Reporting web portal is the main entry point for users, project leaders, mentors, and support members to access job-related monitoring data, according to pre-defined permissions. The role-based access provides different levels of information that comprises: job list tables containing their aggregated performance information, timeline graphs per job for the key performance metrics, node-related monitoring data, a live view of the system (LLview client panel, displayed on Fig. 5), queue information, and access to detailed job reports, including a dynamic HTML and a PDF version.

Besides the Job Reporting infrastructure, LLview also contains an Interactive client, which is a stand-alone application that can run on any Linux/Windows/MacOS system and shows the current state of the monitored system. It includes a colour-coded list of the running jobs and their distribution along with the racks and nodes, as well as a visual representation of the queue state.

The LLview SQLite database files and the scripts for managing the databases, importing monitoring data, and generating data for the web interface are located on service nodes of the cluster. The configuration of the databases is done via configuration files in YAML-format. These configuration files contain not only the definition of the databases and table structures, but also rules to import and to aggregate and store the raw monitoring data into further tables.

In summary, LLview is an efficient tool that will provide a comprehensive monitoring and reporting interface for job-related information on the EUPEX Pilot for system administrators and applications developers alike, with a user-friendly web portal and interactive client, using SQLite databases, with YAML-format configuration files.

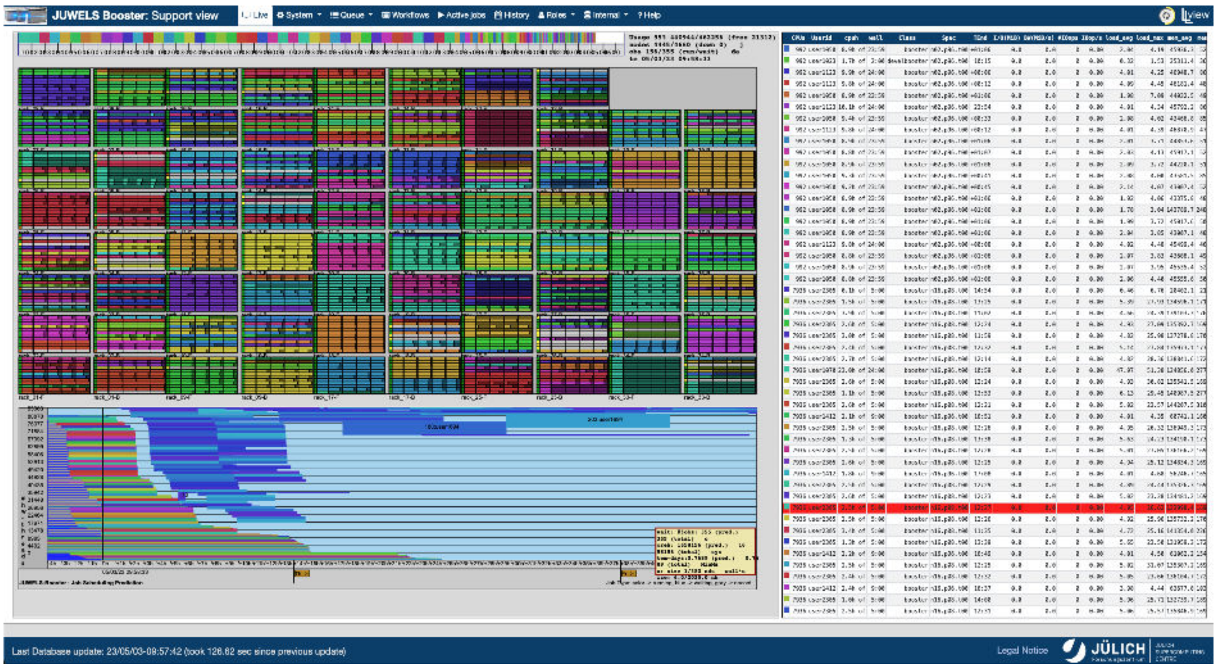

JUWELS Booster: Support view
Live
System
Queue
Workflows
Active jobs
History
Roles
Internal
Help
LLview
Last Database update: 23/05/03-09:57:42 (took 126.82 sec since previous update)
Legal Notice
JÜLICH

FIGURE 5: LIVE VIEW OF THE DEEP SYSTEM IN LLVIEW, DISPLAYING A COLOR-CODED JOB LIST ON THE RIGHT SIDE, THEIR DISTRIBUTION ON THE SYSTEM (BOXES ON THE LEFT) AND ON THE QUEUE (BOTTOM LEFT).

Developments/Extensions planned within EUPEX

LLview is planned to be extended to make it easier to customise, adding new metrics or to adjust what is displayed for the different user roles. Furthermore, the installation and configuration process of LLview will be also made easier. These are important steps to setup and maintain a monitoring infrastructure within EUPEX and other new systems, as well as to integrate relevant metrics.

Timeline for development and for the integration

M18
- Add interactive detailed report
- Extend and simplify of how new metrics are added from “adapters”
- Create new “adapters” to collect metrics from different sources (e. g., Prometheus)

M36
- Create a simplified installation package (to be released Open Source) that can be used to setup the monitoring infrastructure in new systems
- Investigate the possibility of adding the package inside the Ocean project

3 Execution Environment

The execution environment of the EUPEX software stack has to provide a coherent view on the MSA system composed of strongly linked modules while enabling an efficient utilisation of its resources. However, in contrast to the management stack mainly concerned with the demands of the system operators, the execution environment also has to take the requirements of the users and application developers into account. The goal is a flexible and efficient utilisation of the available resources which is enabled by modularity awareness throughout the stack. This is achieved by covering the following areas:

- Programming models for system-wide programming,
- Performance tuning,
- Orchestration, and
- Technologies for the exploitation of hardware capabilities.

3.1 Programming Models

The EUPEX platform aims at supporting a variety of different workloads, ranging from traditional HPC codes with two-sided message-based communication to workflows spanning multiple modules and encompassing different programming paradigms. Therefore, we support a wide range of programming models for system-wide programming, which are introduced in the following sections.

3.1.1 ParaStation MPI

The ParaStation Modulo software suite features a versatile execution environment and runtime called ParaStation MPI. This is especially designed for large-scale supercomputers following the design principles of the MSA. Its node-local resource management component ParaStation Management integrates seamlessly into the Slurm batch queuing system. This integration is facilitated by the psslurm plugin directly interfacing with the central Slurm scheduling daemon (`slurmctld`) by using the native Slurm protocol. In a ParaStation-based setup, the node-local Slurm daemons (i. e., `slurmd` and `slurmstepd`) are replaced by the ParaStation Daemon (psid). All psids on the system form a scalable network of process management daemons responsible for process startup and control, intra-job resource assignment, and the proper cleanup after job termination. ParaStation Management comes with comprehensive support for the PMIx standard—an emerging standard for process and resource management in HPC environments. This way, all parallel runtimes supporting this interface can build upon portable and well-defined access to commonly needed services such as process discovery and endpoint identification.

Therefore, the ParaStation MPI runtime supports PMIx as well, providing the execution of large-scale MPI workloads on modular systems. This is an MPICH derivate (currently based on the MPICH 4.1.1 release) and supports the most recent MPI-4 standard (cf. Fig. 7). Efficient communication in MSA systems is enabled by its low-level, point-to-point communication library pscom. This supports different interconnects and transport protocols common in HPC (e. g.,

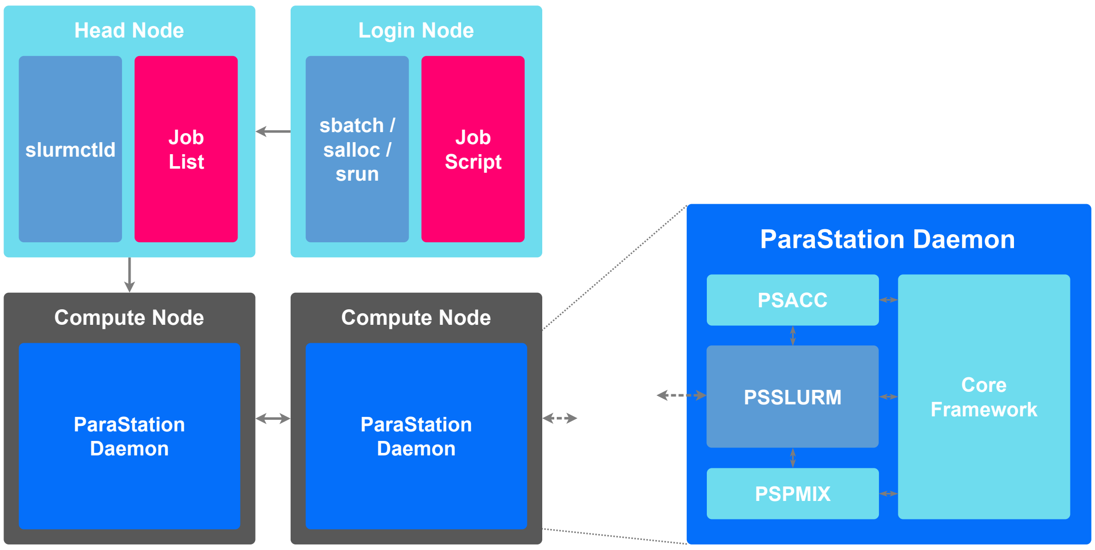


FIGURE 6: PARASTATION MODULO'S PROCESS MANAGEMENT AND ITS INTEGRATION WITH THE SLURM WORKLOAD MANAGER.

InfiniBand (IB), Unified Communication X (UCX), Omni-Path Architecture (OPA), BullSequana eXascale Interconnect (BXI), etc.) concurrently. This way, the computing processes automatically choose the transport promising the best performance depending on their locality within the system. Additionally, the pscom comes with support for network bridging. This even allows for the execution of modular workloads and workflows in case of heterogeneous network landscapes, i. e., the MPI processes are transparently presented a direct, virtual connection even if they are located in different physical networks.

Developments/Extensions planned within EUPEX

Both the ParaStation MPI execution environment and its runtime will be adapted to the particularities of the EUPEX platform. In the first step, compatibility with the compiler toolchain will be ensured to create a first working environment. This is already possible on the EAP. Subsequently, support for the target systems can be added which also includes:

- Pinning strategies,
- Efficient device-to-device communication, and
- MSA support especially regarding the federated network.

Timeline for development and for the integration

M24	ParaStation MPI runtime available on the EAP
M32	Initial support for efficient device-to-device communication via BXI
M42	Final deployment and integration of MPI on the EUPEX platform

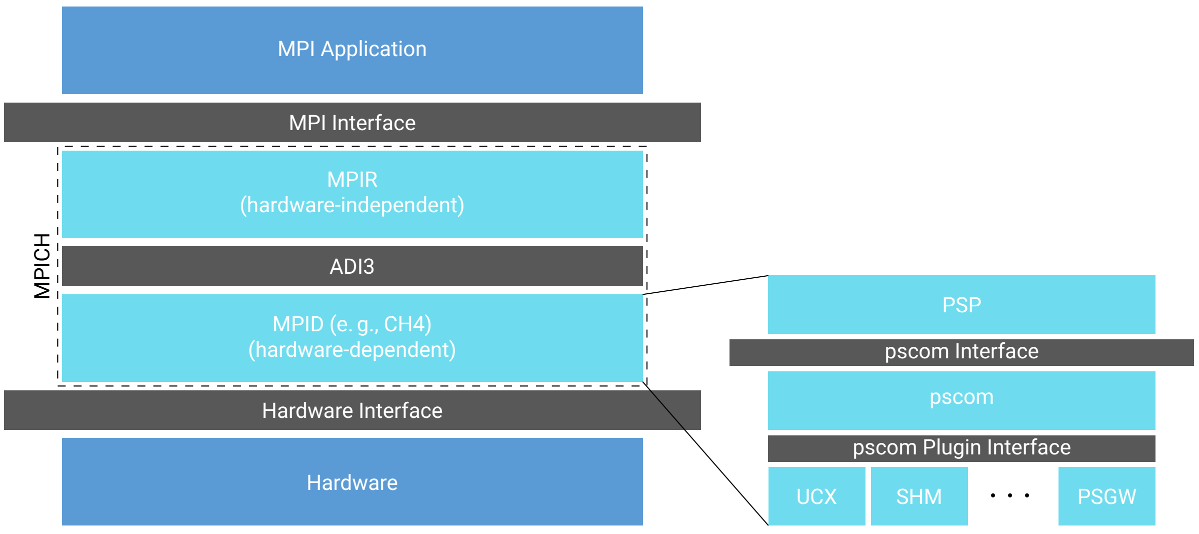


FIGURE 7: THE ARCHITECTURE OF PARASTATION MPI: THE PSCOM IS USED TO IMPLEMENT THE PSP LAYER (B) FOR AN INTEGRATION INTO THE GENERAL MPICH SOFTWARE STACK (A). (BASED ON [12])

3.1.2 Atos Open MPI

Open MPI is an open-source implementation of the MPI standard developed and maintained by consortium of academic and industry partners. As such, Open MPI proposes a production quality software, deployed and used in many-large scale system including several of the TOP500 machines. In its current version 4, Open MPI implements the full MPI 3.1 standard using a modular approach. This approach enables the coexistence of multiple implementations of the same set of functionalities called modules and the selection at runtime of the module to execute. It should be noted that the version 5 of the software is currently developed which implements most of the MPI 4 standard, and is likely to be available in the EUPEX project time frame. Atos proposes an implementation based on the Open MPI open source. The resulting version extends the community one by supporting the BXI interconnect and by implementing either new functionalities such as notified RMA or partitioned collective communications, or by developing new implementation of existing functionalities, such as new methods for achieving collective communications.

Overall Atos Open MPI support multiple interconnects such as IB using UCX or Atos BXI, and it natively relies on PMIx for interfacing with the resource manager. While it already supports the entire MPI-3.1 standard, it will implement MPI-4 specific API such as the MPI sessions in its version 5. Atos Open MPI also provides several implementations of collective communications targeting the MSA by using a hierarchical approach to take into account the topology of the machine. These implementations extend the existing Hierarchical AutotuNed (HAN) component of the open source by providing n-levels topology support for MPI_Bcast, MPI_Gather, MPI_Allgather, MPI_Reduce, MPI_Allreduce and MPI_Alltoall.

Developments/Extensions planned within EUPEX

We will extend Open MPI within EUPEX to:

- Support the EUPEX CPU and GPU
- Provide efficient communication over the last BXI interconnect technology.

Timeline for development and for the integration

M36	Open MPI available on the EAP
M42	Support for efficient device-to-device communication via BXI
M48	Final deployment and integration of Open MPI on the EUPEX platform

3.1.3 GPI-2

GPI-2 [13, 14] is an implementation of the Global Address Space Programming Interface (GASPI) which intends to be an API specification for the Partitioned Global Address Space (PGAS). The main purpose of GASPI is to initiate the transmission from a bulk-synchronous massage-passing based programming paradigm, to one that leverages asynchronous one-sided Remote Direct Memory Access (RDMA) powered communication principles.

Developments/Extensions planned within EUPEX

Goethe University Frankfurt contributes a new communication backend for GPI-2 based on the Portals4 API to provide direct support for the Bull eXascale Interconnect (BXI), i. e., native BXI support for GPI-2. This work will be done in cooperation with the RED-SEA project by providing access to an early prototype system, which is partially equipped with the latest BXI hardware and software.

To be compliant with the overall MSA concept of the EUPEX Pilot system and to provide MSA awareness for the GASPI programming model, Goethe University Frankfurt will collaborate with the Fraunhofer Institute and the DEEP – Software for Exascale Architectures (DEEP-SEA) project to port a pscom-based backend for GPI-2.

In the final phase of the project, the focus will shift to implementing and investigating the usability of direct accelerator communication with GPI-2 over BXI and pscom.

Timeline for development and for the integration

M24	Prototype implementation of the native BXI backend for GPI-2
M36	Open-source release of the BXI backend for GPI-2
M42	Prototype implementation of the pscom backend for GPI-2

3.2 Performance Tuning

3.2.1 ACCO

Atos Collective Communications Optimizer (ACCO) is a tool for tuning Open MPI collective communications. Open MPI includes multiple components for implementing collective communications. The two main components developed by the community are "Tuned", the default component used in Open MPI version 4, and "HAN", the expected default component of Open MPI version 5. Both of these components implement several algorithms for each type of collective communication. The performance of each of these implementations depend among other things on the considered process deployment, interconnect, or message size to exchange. Thus both "Tuned" and "HAN" provide the capability to select which implementation to use with a parameter. Moreover, most of these algorithm implementations can be further tuned using parameters for selecting for instance the segmentation size of messages, or the width of the topological tree. Finding the best set of values for the different parameters for each collective communications on each selected configurations and use cases can lead to better performance for an application. ACCO aims at finding these values by running multiple time a micro-benchmark. It currently provides 2 different methods: either by exploring the entire parameter space, which can be time expensive, or by using black-box optimisation technique such as genetic algorithms.

Developments/Extensions planned within EUPEX

We will extend ACCO within EUPEX to:

- Support the tuning over the EUPEX architecture
- Support the tuning of application's communication when using Open MPI
- Support the tuning of collectives when training deep learning Artificial Intelligence model using Horovod

Timeline for development and for the integration

M36	ACCO available on the EAP
M42	Support tuning applications's communication
M48	Final deployment and integration of ACCO on the EUPEX platform

3.2.2 BHCO

BHCO aims at accelerating MPI + Open Multi-Processing (OpenMP) applications. The main tool included in BHCO targets the progress of MPI communication during an OpenMP computation. When writing a MPI + OpenMP application, it is quite common to decompose this application in such a way that a non blocking MPI communication is called right before entering a computation parallelized with OpenMP with the hope to overlap both communication and computation. In such case, a MPI call is introduced after the completion of the OpenMP section to complete the communication. Unfortunately as both OpenMP and MPI runtime are separated libraries that do not interact with each other, the MPI communication progression while the application has entered OpenMP cannot be ensured and the transfer actually happens in the completion call, thus inducing no real overlap. A possible way to solve this issue consists in keeping a MPI thread running which will progress this communication, but which also will interfere with the computing OpenMP threads and thus impact the performance. With BHCO, using OMPT, a progression task will be called by an OpenMP thread that reached the end of its computation while waiting for the other threads to finish their computation. This progress task is specific to the Open MPI implementation. As this is done through OMPT, there is no need to modify the code of an application to benefit from this solution.

Developments/Extensions planned within EUPEX

We will extend BHCO within EUPEX to support the new EUPEX CPU

Timeline for development and for the integration

M36 > BHCO available on the EAP

M48 > Final deployment and integration of BHCO on the EUPEX platform

3.2.3 libVC

libVersioningCompiler [15] a dynamic optimisation library designed to reduce the HPC expertise skill gap required to build efficient HPC applications, by providing a simple and efficient way to dynamically optimise kernels. It provides a set of APIs to perform runtime optimisations on C/C++ code. libVersioningCompiler is organised in the following main high-level components: **Version**, which represents a compute kernel to provide symbols ready to be executed; **Option**, which represents each of the flags and parameters that are passed to libVersioningCompiler in order to compile a version of a computing kernel. **Compiler**, which defines the interface that compiler implementations must comply with to be used with libVersioningCompiler. **Utils**, which includes a set of utility functions that are being used to have a ready-to-use library. The libVersioningCompiler library was initially developed for LLVM 6 as part of the ANTAREX FETHPC project. In EUPEX, it was updated to support the latest three major releases of LLVM (13, 14, and 15). The code has been further updated to leverage the C++17 standard. The original work libVersioningCompiler was intended to work with C/C++ kernels only. The EUPEX version now supports kernels in any language that provides ABI compatibility with C (for EUPEX and, more in general, HPC workloads, this is particularly important as the extended version is now able to work with Fortran kernels). Finally, to address those HPC use cases, we have developed an implementation of libVersioningCompiler that leverages a distributed network to fetch optimised kernels when available. The primary goal of this implementation is to provide easy-to-scale optimisation capabilities as needed. For example, a small and scalable set of computing nodes may be used to compile optimised versions of the kernel, allowing the rest of the nodes to work with the optimised version of the function. Our implementation is designed to be extremely flexible, allowing users to customise the network stack for both the DHT and the serving/download process. To achieve this, we have implemented the download mechanism using ZeroMQ, which provides fast communication channels and supports multiple network stacks. To allow the distribution of available versions' sockets, we use a distributed hash table (DHT). Its structure allows the coordination of multiple code producers and consumers, while remaining highly flexible with nodes joining and leaving the network. The DHT network we are using is based on a Kademlia DHT implementation called OpenDHT. To drastically reduce the overhead caused by the openDHT setup and configuration, our implementation uses elliptic curve (EC) based cryptographic identities. All the proposed extensions have been released to the original open source project.[1] EUPEX application codes can therefore leverage libVersioningCompiler to generate and run multiple versions of a given kernel, with different optimization options. An example of integration for a given function `FUNCTION_NAME` defined in file `FILENAME_SRC` is provided herebelow, using the libVersioningCompiler high-level interface.

We need to include the libVersioningCompiler header file:

```
#include "versioningCompiler/Utils.hpp"
```

Then, we initialize the library by calling once, on setup, the following method:

```
vc::vc_utils_init();
```

Once the library is set up, for every desired Version object, we can create the corresponding compiled code:

```
vc::version_ptr_t v = vc::createVersion(FILENAME_SRC,
                              FUNCTION_NAME,
                              {vc::Option("O", "-O", "2"), ... }
                             );
signature_t fn_ptr = (signature_t) vc::compileAndGetSymbol(v);
if (fn_ptr) { // check if correctly compiled and loaded symbol
  fn_ptr(42); // run the compiled function version
}
```

It is also possible to perform split-compilation, by precompiling the LLVM-IR, as well as other complex compilation policies by leveraging a finer-grained API that is described in full in the libVersioningCompiler documentation. The abovedescribed high-level API is however sufficient for the majority of use cases.

3.3 Orchestration

3.3.1 StreamFlow

StreamFlow [16] is a Workflow Management System (WMS) able to orchestrate topology-aware workflows on top of hybrid and heterogeneous infrastructures (e. g., cloud+HPC or MSA). Indeed, StreamFlow has been designed around two main principles. First, it supports task executions in multi-container environments, targeting complex microservices architectures in the cloud. Second, it relaxes the requirement of a single shared data space, automatising cross-stack data movements over hybrid infrastructures. StreamFlow exposes a fully declarative interface to model hybrid workflows. In particular, it relies on the Common Workflow Language (CWL) open

[1]https://github.com/skeru/libVersioningCompiler

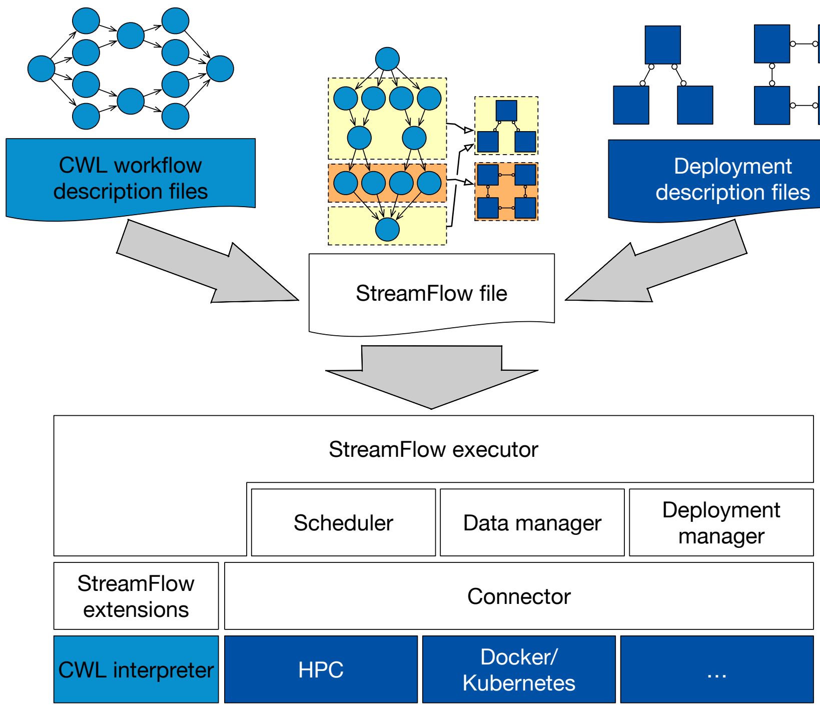


FIGURE 8: STREAMFLOW FRAMEWORK'S LOGICAL STACK

standard for describing steps and their dependencies and on a YAML-based configuration file for defining the available execution infrastructures and binding each step to a proper execution location. A declarative approach allows a clear separation between the host code (e. g., an MPI application or a TensorFlow script), the coordination between different steps, and their offloading to different infrastructures (e. g., a Kubernetes-based cloud environment or a Slurm-managed HPC facility). Plus, the StreamFlow plugin system enables a straightforward introduction of new deployment methods and seamlessly porting existing workflows onto them.

This task aims to integrate MSAs as first-class citizens in the StreamFlow ecosystem, making it a proper tool for rapid prototyping workflows of legacy applications onto the MSA infrastructure. In detail, we contribute a new StreamFlow plugin capable of exploiting MSA awareness to improve the performance of heterogeneous HPC workflows. This approach has two main advantages. First, users can evaluate MSAs without modifying and recompiling the host code. Second, application-specific and architecture-specific optimisations can be introduced at the plugin level without modifying the StreamFlow core codebase. Plus, introducing MSAs in the StreamFlow ecosystem makes it straightforward to evaluate their usage in combination with more standard execution infrastructures, e. g., multi-MSAs or cloud+MSA. With these features, StreamFlow will become an integral part of the EUPEX software stack, providing an efficient and innovative way to manage workflows in large-scale MSA systems. At the same time, integrating other tools in the stack will bring even more benefits in terms of expressive power and performance optimisation. For example, the integration with CAPIO (see Sec. 5.1.1) will enable transparent streaming I/O capabilities in distributed workflows.

Timeline for development and for the integration

M18
- Open source release of StreamFlow 0.2.0, with enhanced support for third-party plugins and more flexible configuration of HPC execution environments;
- Collection of application requirements in terms of workflow patterns and performance optimisation.

M36
- Open source release of the StreamFlow MSA plugin, fully integrated in the EUPEX software stack;
- Open source release of the StreamFlow integration with CAPIO.

3.4 Exploitation of Hardware Capabilities

The performance of modern computing platforms comes with an increasing hardware complexity. Many resources of many different kinds are used, with complex interconnection between them, making data movement an expensive constraint. Hence it is important to optimise the software so as to properly exploit all the capabilities of the hardware. We describe in this section how several software components will contribute to a better utilisation of the EUPEX pilot.

3.4.1 xHC

Modern HPC nodes comprise high core counts and are able to host tens to hundreds of processes. Moreover, they contain elaborate internal topologies, consisting of multiple sockets, NUMA domains and cache hierarchies. As a result, communication between different pairs of cores takes place over paths of different costs (e. g., over a shared last level cache, inside or outside a NUMA node, between different sockets). Thus, the performance of collectives at the intra-node level is level is crucial, but faces several challenges. In this task, we contribute a new component for collectives in OpenMPI: XPMEM-based Hierarchical Collectives (XHC). XHC provides optimised intra-node collectives to ad-dress these challenges of modern nodes. Within the scope of the project, XHC will provide an im-plementation for at least one indicative operation from each category of collectives: broadcast (one-to-all), reduce (all-to-one), and allreduce & barrier (all-to-all). All collective primitives offered by XHC share the following characteristics: (a) topology awareness, (b) single-copy or shared-memory-based transfers, (c) direct implementation over memory, i. e., not based on the point-to-point layer, and optionally (d) cache coherency protocol awareness. Several optimisations for improving the performance of baseline primitive implementations are planned, including:

- Uniformly distributing reduce and allreduce operation load among all available ranks.
- Using dynamic leadership to mitigate the effects of late rank arrivals.
- Advanced tuning of algorithms and features, across different levels of the hierarchy.
- Examine optimised access patterns for shared control structures.
- Explore the benefits of algorithms combining both flat and hierarchical communication patterns for different phases of the same operation, for all-to-all primitives.

XHC focuses on and only handles the intra-node aspect of collectives. For the inter-node part, our goal is to integrate XHC with OpenMPI's HAN. This requires the introduction of various changes and optimisations on either of the two sides, in order to achieve seamless and efficient operation.

Timeline for development and for the integration

M18 Open source release of XHC v2: https://eupex.eu/sw-environment/sw-environment-execution-environment

- Optimisations for collectives
- Analysis of cache coherence protocol implications

M36 Open source release of XHC v3

- Optimisations for primitives and performance profiling on arm64
- Integration and evaluation of XHC+HAN

3.4.2 hwloc (Portable Hardware Locality)

The open-source project hwloc [17, 18] is the de-facto standard tool for discovering the resources in computing platforms (processors, cores, NUMA nodes, I/O devices) as well as their organisation and locality (which resources are close or far away from each other, which memory is local to which CPUs etc). This knowledge is critical to HPC nowadays because because the placement tasks and buffers has a critical impact on performance on complex architectures (cores must be close the data they manipulate, to resources to they work with, etc). In the context of this project, several software layers will rely on hwloc. For instance MPI implementations (Section 3.1.2) and the XHC component (Section 3.4.1) will get information on the machine's hardware topology and memory hierarchy from hwloc.

hwloc already supports a long list of platforms and architectures, including some ARM platforms, but hardware availability is limited. The diversity of ARM implementations sometimes require updates in the hwloc code. hwloc also supports heterogeneous memory [19, 20]. However this kind of heterogeneity is still rare and was only recently supported by the ACPI hardware specification and the Linux kernel. Hence some hwloc updates might be required when new implementation such as the EUPEX pilot are tested.

hwloc gathers the relevant information from the operating system APIs and hardware tables. Hence one critical part of our work in this project is first to make sure the Linux kernel and the ARM firmware expose extensive (and correct) hardware information. We are working towards this goal with ARM vendors, especially regarding ACPI tables. For instance, exposing the PPTT table is required for CPU topology while the HMAT table is required for heterogeneous memory information. Once this information is populated in the firmware, a recent Linux kernel will expose it to user-space and existing hwloc releases should be able to expose the corresponding information to runtimes and applications. If the information is missing in hardware, hwloc still has the ability to hardwire information for a specific platform.

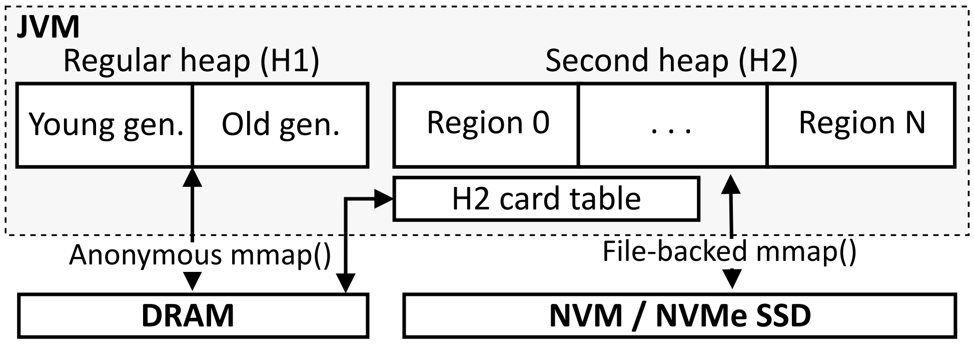


FIGURE 9: TERAHEAP DESIGN OVERVIEW

Timeline for development and for the integration

M18 > Release 2.9 of hwloc is compatible with the most popular ARM platforms and has support for heterogeneous memory.

M36 > Release 3.x of hwloc exposing all CPU and memory details of the EUPEX pilot hardware.

3.4.3 TeraHeap

TeraHeap, is a system that eliminates Serialisation/Deserialisation overhead and expensive garbage collection in managed runtime systems, such as the JVM, for applications that require large address spaces. TeraHeap relies on three concepts. (1) It eliminates S/D cost by extending the managed runtime (JVM) to use a second high-capacity heap (H2) over a fast storage device. (2) It offers a simple hint-based interface, allowing big data analytics frameworks to leverage knowledge about objects to populate H2. (3) It reduces GC cost by fencing the garbage collector from scanning H2 objects while maintaining the illusion of a single managed heap. We implement TeraHeap in OpenJDK and evaluate it with 15 widely used applications in two real-world big data frameworks (Spark and Giraph). Our evaluation shows that for the same DRAM size, TeraHeap improves performance compared to native execution, while it provides similar performance when using significantly less DRAM.

Timeline

M18 > First version of TeraHeap is available as open source: https://eupex.eu/sw-environment/sw-environment-execution-environment

M36 > Porting to Java 17 and optimisations.

3.4.4 KNoC/Knot

Cloud and HPC increasingly converge in hardware platform capabilities and specifications, nevertheless still largely differ in the software stack and how it manages available resources. The HPC world typically favours Slurm for job scheduling, whereas Cloud deployments rely on Kubernetes to orchestrate container instances across nodes. Running hybrid workloads is possible by using bridging mechanisms that submit jobs from one environment to the other. KNoC (Kubernetes Node on Cluster) is a virtual node (kubelet) for Kubernetes that transparently manages the container lifecycle on a remote HPC cluster using Slurm for scheduling and Singularity/Apptainer as the container runtime. In the initial implementation, the Kubernetes installation hosting the KNoC node may run locally to the user (in a simple virtual machine), or even in a "sidecar" environment offered by the HPC centre. In the second iteration of KNoC, we explore a design that runs all Kubernetes components (including the special kubelet) within HPC. KNoC provides users with an effective way to utilise resources by a combination of well-known tools, APIs, and more interactive and user-friendly interfaces as is common practice in the Cloud domain, as well as seamlessly combine Cloud-native tools with HPC jobs in converged, containerised workflows. To simplify the deployment of such Cloud-native components we use Knot. Knot is a full-featured software stack for facilitating data science in Kubernetes, by integrating a web-based environment that includes an extensible set of productivity tools and services. At its core, the Knot dashboard supplies the landing page for users, allowing them to design workflows (using the popular Argo Workflows language), and specify execution parameters through a user-friendly inter-face.

Timeline

M18	First version of KNoC/Knot is available as open source: https://eupex.eu/sw-environment/sw-environment-execution-environment
M36	Second version of KNoC with support for containerised workflows on HPC

4 Tools for Performance and Energy Efficiency

Power-management, energy-efficiency, and performance optimisation are key aspects to account for on the road to Exascale. The software stack needs to provide tools for the application developers and the system-manager to (i) monitor and profile the power/energy consumption, and the performance bottlenecks of the application runs; and (ii) when the application faces performance bottlenecks to leverage the HW power management knobs to reduce energy consumption automatically, and to support the programmer for semi-automatically tuning of the application kernels and power-management knobs. Moreover, the software stack needs to be adapted to the EUPEX Pilot innovations and characteristics, namely: the Modular System Architecture (MSA), the Sipearl Rhea architecture and power management knobs, and the application characteristics. It will also integrate with the best-in-class European technologies developed in the H2020 FETHPC and EuroHPC01 projects.

4.1 Energy monitoring and power management

4.1.1 BEO

BEO (Bull Energy optimizer) is a tool for monitoring the power, energy, temperature and performance of the whole cluster infrastructure. Based on out-of-band monitoring through standard protocols (IPMI, SNMP, Redfish) and in-band monitoring (BDPO) and on a consolidated and distributed database, BEO can provide energy related insights without instrumentation. Connected with a system manager, BEO can provide detailed accounting metrics for a given job. BEO is able to set a power limit (power capping) for the whole cluster infrastructure or a subset of the cluster. BEO will ensure that the limit will not be overpassed. BEO can be easily extended to support any additional hardware. BEO contains secured (OpenID connect, LDAP compatible) interfaces: CLI, Rest API (OpenAPI specification) and customisable web user interface intended for system administrators and end users.

Timeline

M36
- Full support of SiPearl Rhea nodes (metrics and Power capping).
- Power capping feature adaptation to take the benefit from low level firmware services developed in WP4 by CINI (Pulp – Specific HW card to manage Power).

4.2 Energy efficiency runtime systems

4.2.1 MERIC

The open-source (BSD-3 licence) runtime system MERIC [21] developed by IT4Innovations is designed to minimise the energy consumption of the HPC infrastructure executing a parallel application by dynamic tuning a wide range of hardware knobs. The methodology of dynamic tuning comes from the Horizon 2020 project READEX [22] under which the development of the MERIC has started. The library and associated tools perform a detailed analysis of complex application behaviour, identification of the optimal hardware settings concerning energy consumption and runtime, and dynamic tuning during the application runtime. The output of an application analysis is possible to visualise using the RADAR visualiser [23].

In the reported period of the EUPEX project, support for A64FX energy measurement was implemented to provide energy-efficiency analysis of applications executed on the EUPEX Alfa Pilot platform (EAP). We have identified a list of power management knobs that A64FX suppose to implement, based on publications of RIKEN experts [24]. Unfortunately, support for tuning these knobs has not been implemented, since the CEA Irene administrators have not found a safe way how to expose them to userspace. However, Rhea will use a different power management interface (SCMI [25]) and therefore this limitation is not critical. Additionally, EUPEX applications can be tested for usage with MERIC on EUPEX in-kind systems that provide according hardware support (e. g., the Karolina EuroHPC Petascale system).

Timeline

M24
- EUPEX Alpha Pilot platform user guide for energy measurement using MERIC.
- MERIC-COUNTDOWN cooperation: providing energy measurement performed by MERIC to COUNTDOWN.
- Implement MERIC support for SCMI power governor used by Rhea CPU.

4.2.2 COUNTDOWN

COUNTDOWN is a methodology and a tool for identifying and automatically reducing the frequency of the computing elements in order to save energy during communication and synchronisation primitives. COUNTDOWN is able to filter out phases which would detriment the time to solution of the application transparently to the user, without touching the application code nor requiring recompilation of the application. Besides its primary use as an energy-saving framework, COUNTDOWN can be a powerful monitoring tool as it allows us to track and record low-level commands of the parallel application. This then allows a granular monitoring and performance analysis of the application running on specific hardware.

In EUPEX, COUNTDOWN will have two principal uses: 1) it will be used as an energy-saving framework which reduces the CPU frequency during the communication and synchronisation phases of the parallel application, and 2) it will be used as a granular monitoring tool that will give us more insights into the performance of applications running on pilot hardware.

Timeline

M24
- EUPEX Alpha Pilot platform user guide for COUNTDOWN.
- Integrate COUNTDOWN-MERIC cooperation: integrate in COUNTDOWN energy measurement collected by MERIC.

M36 › Implement COUNTDOWN support for SCMI power governor used by Rhea CPU.

4.2.3 BDPO

BDPO is a lightweight daemon executed on each compute node which leverages fine-grain monitoring of CPU-centric metrics such as the IPC (number of retired instruction per reference clock cycle of the processor) to determine if the executed parallel application requires a lot of computational power or if it is partially, nay heavily, stalling. When such a portion of execution, which is called a "phase" afterwards, is detected, BDPO lowers the frequency and voltage of the processors on-the-fly, which notably allows to decrease their power consumption at the cost of some performance degradation (the more stalling the processors, the more negligible the induced performance degradation). When detecting a compute-intensive phase, BDPO resets the voltage and the frequency of the processors to higher values, if necessary. By doing so, BDPO aims at optimising the energy-efficiency associated with the execution of HPC applications by leveraging their stall-intensive phases, while having an impact as insignificant as possible on the associated performance.

Timeline

M24 › Integration of BDPO with the EUPEX's selected job scheduler for automatic start and stop.

M36 › Integration with MERIC especially for user-marked regions of interest.

4.3 Performance Tools

4.3.1 Score-P

The Score-P measurement infrastructure [26, 27] is a highly scalable and easy-to-use tool suite for profiling and event tracing of HPC applications. Score-P offers the user a maximum of convenience by supporting a number of analysis tools. Currently, it works with Cube, Scalasca, Vampir [28], Tau [29], and Extra-P [30] and is open for other tools. Score-P comes together with the new Open Trace Format Version 2 [31], the Cube4 profiling format [32] and the Opari2 instrumenter for OpenMP applications. Score-P instrumented applications capture events from various sources (MPI, threads, GPUs, I/O, source code, interrupts, Hardware counters [33]), augment them with timestamps and hardware counters before passing them to profiling and tracing backends (see Figure 10).

Score-P is available under the 3-clause BSD Open Source license [34].

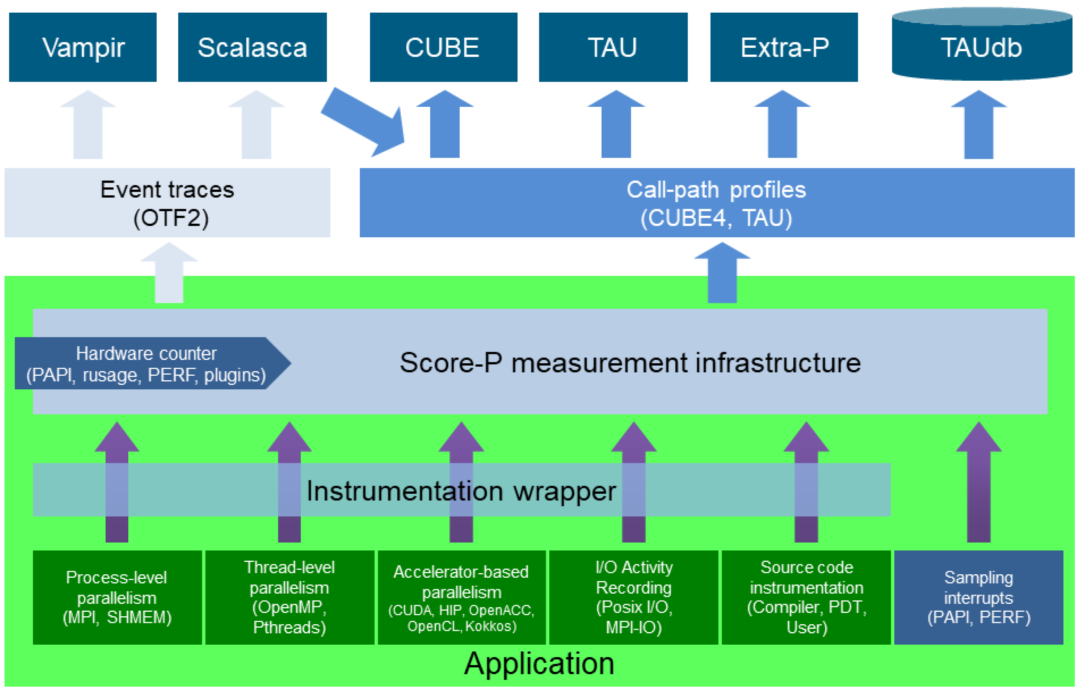

Vampir
Scalasca
CUBE
TAU
Extra-P
TAUdb
Event traces (OTF2)
Call-path profiles (CUBE4, TAU)
Hardware counter (PAPI, rusage, PERF, plugins)
Score-P measurement infrastructure
Instrumentation wrapper
Process-level parallelism (MPI, SHMEM)
Thread-level parallelism (OpenMP, Pthreads)
Accelerator-based parallelism (CUDA, HIP, OpenACC, OpenCL, Kokkos)
I/O Activity Recording (Posix I/O, MPI-IO)
Source code instrumentation (Compiler, PDT, User)
Sampling interrupts (PAPI, PERF)
Application

Figure 10: Score-P component architecture

4.3.2 Scalasca

The Scalasca Trace Tools [35, 36, 37] are a collection of trace-based performance analysis tools that have been specifically designed for use on large-scale systems featuring hundreds of thousands of CPU cores, but also suitable for smaller HPC platforms. A distinctive feature of the Scalasca Trace Tools is its scalable automatic trace-analysis component which provides the ability to identify wait states that occur, for example, as a result of unevenly distributed workloads. Especially when trying to scale communication intensive applications to large process counts, such wait states can present severe challenges to achieving good performance. Besides merely identifying wait states, the trace analyser is also able to pinpoint their root causes (i. e., delays) [38], and to identify the activities on the critical path of the target application [39], highlighting those routines which determine the length of the program execution and therefore constitute the best candidates for optimisation.

Scalasca is available under the 3-clause BSD Open Source license [40].

Timeline

M24 › Score-P and Scalasca ported to EUPEX Alpha Pilot platform and adapted to platform compiler, MPI libraries, and power hardware counter interface.

- EUPEX Alpha Pilot platform user guide for performance analysis using Score-P and Scalasca.

M48
- Score-P and Scalasca ported to EUPEX Pilot platform and adapted to platform compiler and MPI libraries.
- Score-P and Scalasca integrated in EUPEX Software Stack

4.3.3 Cube

The Cube Software Framework [41] facilitates the collection, processing, and analysis of application performance profiling data from Score-P and Scalasca. With this framework, users have access to a comprehensive set of tools for measuring and understanding application performance.

Cube (CUBE Uniform Behavioral Encoding) is suitable for analysing a wide variety of performance data for parallel programs including MPI and OpenMP applications. Cube has been designed around a high-level data model of program behaviour called the cube performance space. The Cube performance space consists of three dimensions: a metric dimension, a program dimension, and a system dimension. The metric dimension contains a set of metrics, such as communication time or cache misses. The program dimension contains the program's call-tree, which includes all the call paths onto which metric values can be mapped. The system dimension contains the components executing in parallel, which can be processes or threads depending on the parallel programming model. Each point of the space can be mapped onto a number representing the actual measurement for metric while the control flow of process/thread was executing call path. This mapping is called the severity of the performance space.

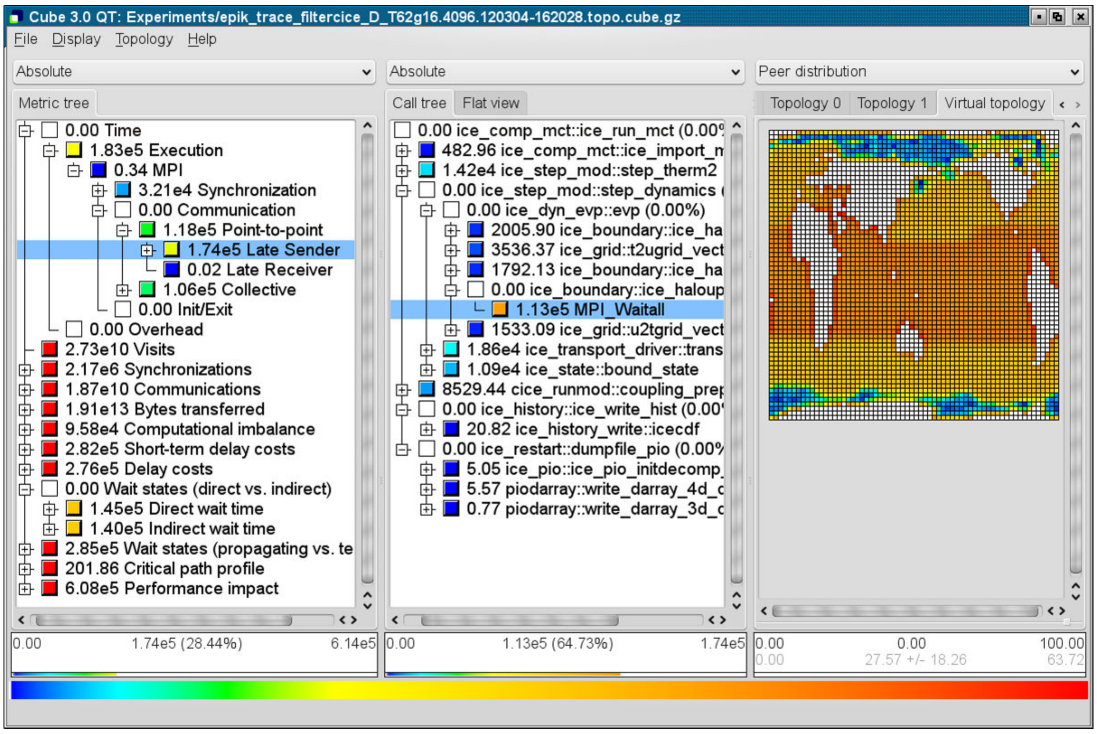
The display component CubeGUI can load such a file and display the different dimensions of the performance space using three coupled tree browsers. The browsers are connected in such a way that you can view one dimension with respect to another dimension. For an example, see Figure 11.

Cube is available under the 3-clause BSD Open Source license [42].

Timeline

M24
- Cube ported to EUPEX Alpha Pilot platform and adapted to platform compiler.

M48
- Cube ported to EUPEX Pilot platform and adapted to platform compiler.
- Cube integrated in EUPEX Software Stack


```
Cube 3.0 QT: Experiments/epik_trace_filtercice_D_T62g16.4096.120304-162028.topo.cube.gz
File  Display  Topology  Help

Absolute                              Absolute                              Peer distribution
Metric tree                           Call tree   Flat view                 Topology 0   Topology 1   Virtual topology

0.00 Time                             0.00 ice_comp_mct::ice_run_mct (0.00%
  1.83e5 Execution                      482.96 ice_comp_mct::ice_import_m
    0.34 MPI                            1.42e4 ice_step_mod::step_therm2
      3.21e4 Synchronization            0.00 ice_step_mod::step_dynamics (
      0.00 Communication                  0.00 ice_dyn_evp::evp (0.00%)
        1.18e5 Point-to-point               2005.90 ice_boundary::ice_ha
          1.74e5 Late Sender                3536.37 ice_grid::t2ugrid_vect
          0.02 Late Receiver                1792.13 ice_boundary::ice_ha
        1.06e5 Collective                   0.00 ice_boundary::ice_haloup
      0.00 Init/Exit                          1.13e5 MPI_Waitall
    0.00 Overhead                           1533.09 ice_grid::u2tgrid_vect
2.73e10 Visits                            1.86e4 ice_transport_driver::trans
2.17e6 Synchronizations                   1.09e4 ice_state::bound_state
1.87e10 Communications                  8529.44 cice_runmod::coupling_prep
1.91e13 Bytes transferred               0.00 ice_history::ice_write_hist (0.00
9.58e4 Computational imbalance            20.82 ice_history_write::icecdf
2.82e5 Short-term delay costs           0.00 ice_restart::dumpfile_pio (0.00%
2.76e5 Delay costs                        5.05 ice_pio::ice_pio_initdecomp
0.00 Wait states (direct vs. indirect)    5.57 piodarray::write_darray_4d_c
  1.45e5 Direct wait time                 0.77 piodarray::write_darray_3d_c
  1.40e5 Indirect wait time
2.85e5 Wait states (propagating vs. te
201.86 Critical path profile
6.08e5 Performance impact

0.00   1.74e5 (28.44%)   6.14e5      0.00   1.13e5 (64.73%)   1.74e5      0.00          0.00           100.00
                                                                          0.00   27.57 +/- 18.26      63.72
```

Figure 11: CubeGUI result display of a measurement of the CESM sea ice code executing on 4096 processors.

4.3.4 ParaStation MPI

The ParaStation Management will be used to collect and forward node-local accounting information such as performance and energy counters. The main daemon psid is enriched by various plugins extending its functionality. The accounting facility in psid is provided by the psaccount plugin. psaccount is responsible for collecting and aggregating accounting data. The flexible design of psaccount allows for easy integration of new data sources on the compute nodes. The data collected from a compute node will be forwarded to the mother superior and further aggregated. The psslurm plugin will fetch the collected data for each job and step from psaccount and re-package it in the corresponding Slurm RPCs, which will be send to the Slurm scheduling daemon (`slurmctld`).

4.4 Hardware requirements on the EUPEX Protopype

- Exposing hardware performance counters ideally via PAPI and/or perfctr.
- Energy consumption measurement exposed to userspace, 1 kHz and higher power sampling frequency preferred.
- Write permission on relevant hardware power management interfaces.

4.4.1 Energy tools specification initiative

Following the outcomes of the face to face meeting in Zagreb mid February 2023, an initiative to create a working group for energy tools specification has materialised. Composed of volunteers from several WPs (4, 5, 6), the group aims to define the contours and specifications of a suite of tools to accurately report the energy consumption of the application phases and components of the supercomputer, but also to implement the means to undertake an optimisation process. In order to achieve this, it is essential to determine upstream the functionalities and the nature of the information to be implemented on the pilot system. This concerns in particular a way of exposing hardware performance information for collection and the functionalities of components steering. The objective of the working group is to define as precisely and realistically (i. e., achievable in the EUPEX project context) as possible, the specifications of the pilot system in this domain.

5 I/O Stack

The role of T5.4 is to provide an advanced I/O software stack for the EUPEX Pilot, which meets the requirements of the Modular Supercomputing Architecture (MSA) [1]. In addition to giving access to files through a shared file system, many middlewares will provide applications and workflows with a tiered storage environment offering higher level features.

The components of the I/O stack are split into 4 categories:

- An In-Memory tier, providing data coupling in workflows
- A high performance tier, providing ephemeral I/O services;
- A set of application specific interfaces,
- A set of tools and interfaces, enabling in depth analysis of the applications and workflows I/O behaviour, and I/O optimisation libraries

The main effort in T5.4 will be about the integration of these bits into a coherent and efficient I/O stack for the EUPEX Pilot. The 4 next sections contain a description of the components in each category.

5.1 In-Memory Tier

5.1.1 CAPIO

CAPIO (Cross-Application Programmable I/O[1]) is a user-space middleware capable of transparently injecting I/O streaming capabilities into file-based token-pushing workflows that communicate through files without modifying their original code. The primary objective of CAPIO is to boost the performance of existing scientific workflows by improving the computation-I/O overlap without modifying the business code of workflow components.

As sketched in Figure 12, the CAPIO middleware is logically composed of two main software layers:

- At a higher level, a *coordination model* that allows the user to express relaxed token synchronisation semantics between producer-consumer steps of a workflow through an *I/O coordination language* (currently leveraging the JSON format) describing when a file is "*fireable*" (i. e., when its content can be accessed) and when a file is "*committed*" (i. e., when it is completed, and no more I/O operations will be performed on it).
- At a lower level, CAPIO provides a *runtime system* composed of a set of per-node user-space servers implementing the distributed data and metadata storage for the files, and an intercept library that must be dynamically linked (through the `LD_PRELOAD` environment variable) to the application steps to intercept file system POSIX system calls executed on a pre-defined CAPIO directory (i. e., the CAPIO mount point for the local node).

[1]https://github.com/alpha-unito/capio

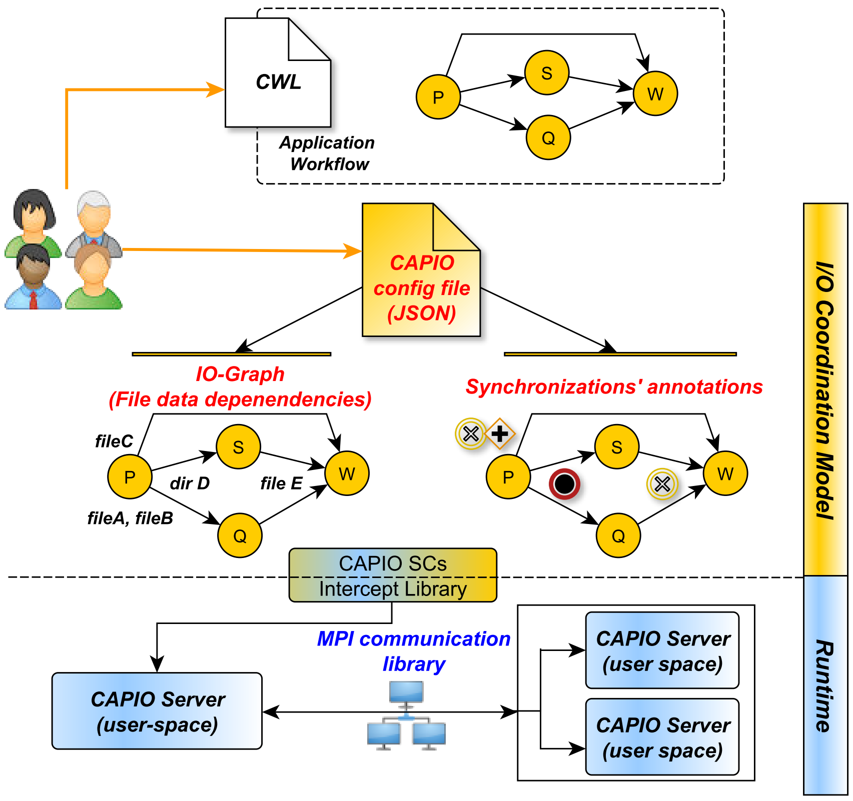

CWL
Application Workflow
P
S
Q
W
CAPIO config file (JSON)
IO-Graph (File data depenendencies)
fileC
dir D
file E
fileA, fileB
Synchronizations' annotations
I/O Coordination Model
CAPIO SCs Intercept Library
MPI communication library
CAPIO Server (user-space)
CAPIO Server (user space)
CAPIO Server (user space)
Runtime

Figure 12: Logical architecture of the CAPIO middleware.

The CAPIO coordination model allows the user to write a configuration file enabling the CAPIO runtime to optimise file data movements between workflow steps and transforming their execution from a batch execution to a streaming one according to the annotated semantics provided by the user through the configuration file.

5.1.2 Damaris

Damaris is a middleware for asynchronous I/O, data analytics and visualisation, targeting large-scale MPI-based HPC simulations. It enables the use of dedicate cores or dedicated nodes for asynchronous I/O and has an emphasis on ease of integration in existing simulations through a simple API, efficient resource usage (with the use of shared memory) and simplicity of extension through plug-ins.

Typical HPC simulations loop through a series of iterations, generating a large volume of data at the end of each iteration and as a result they trigger a heavy I/O burst at the end of each iteration. In the usual approach, output data is saved and are then transferred to an auxiliary post-processing platform for analysis and visualisation. The mentioned data transfer is costly, and no output is available until the end of the post-processing analysis and visualisation phase. The burst I/O pattern can lead to variability in performance and execution time, known as jitter. Damaris mitigates these issues by allowing asynchronous data I/O or processing on dedicated hardware as the data is produced. This kind of processing is termed in situ processing (for processing on dedicated in-node resources) or in transit processing (for processing on separate nodes). Damaris has been shown to reduce jitter and hide latency due to I/O, which leads to improved scalability of computational codes.

Available plugins

CAPIO comes with the following set of plugins, that can be optionally compiled, serving various processing requirements:

- HDF5 output in file-per-core and collective mode (using MPI-I/O);
- Python based analytics;
- Paraview in situ visualisation;
- VisIt in situ visualisation;
- Ascent in situ visualisation (currently through the Python interface).

Software and runtime constraints

For use of Damaris, an application must have a few prerequisites and constraints placed on its runtime. Damaris requires that an HPC simulation is able to specify the use of a different MPI communicator than `MPI_COMM_WORLD` for its MPI based communications. The number of simulation ranks has to be perfectly divisible by the number of Damaris ranks (per node), or, the number of simulation nodes has to be perfectly divisible by the number of Damaris dedicated nodes. For improved runtime the time taken for I/O processing should be shorter than the time between I/O output stages.

Timeline

M18 Compilation and test suite on ARM A64FX

- Development of testing suite covering Damaris dependencies.
- Present Damaris to potential WP3 use cases.
- Using EuroHPC ACROSS project use-case (OPM Flow) as benchmark.

M36 Running on Rhea nodes.

- Compilation and test suite execution.
- Integration of Damaris API with selected WP3 use case (if one found)
- Benchmarking using OPM Flow.

5.2 High Performance Tier

The High Performance tier is based upon specific "data nodes" inserted in the MSA architecture to serve as a fast data storage space for applications and workflows. Data nodes are connected to one or more modules on their high speed interconnect, and contain enough NVMe disks to have a read and write I/O bandwidth similar to the interconnect in order to be non blocking. *Ephemeral I/O services* will be launched upon request to enable data exchanges between workflow steps without involving long term storage.

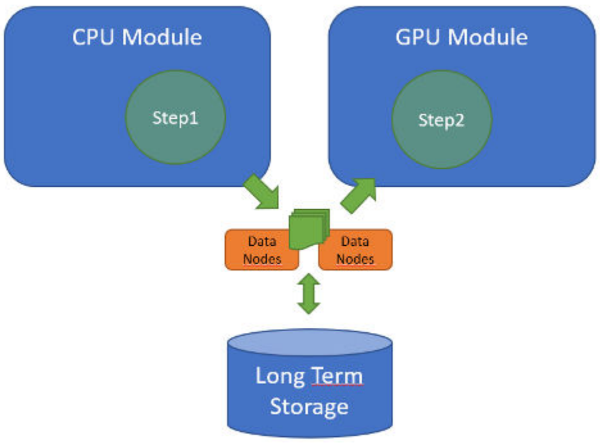


FIGURE 13: MSA AND DATA NODES

5.2.1 KV Store

FORTH's KV store consists of two subsystems: Parallax [43] and Tebis [44].

- Parallax is an LSM KV storage engine that increases system efficiency by categorizing KV pairs according to their size into three size-based categories (small, medium, and large) and applying different policies in each category.
- Tebis (Figure 14) is a distributed Key Value (KV) store designed for flash storage devices (NVMe SSDs) and RDMA networks. Tebis consists of a highly available metadata server and data servers, as shown in Figure 1. Each data server organises the data stored in its flash devices using the Parallax KV storage engine. The metadata server keeps the key ranges that each data server is responsible for and monitors the health of data servers. In case of a failure, the metadata server starts the recovery protocol of the system to a) reassign regions of the faulty data server and b) create extra replicas for recovery purposes.

Placement

The KV store can be deployed either in the storage nodes or the compute nodes to offer a KV abstraction over shared or private storage devices. Note that even when the KV store runs on the compute nodes, it still offers a global storage address space that is accessible from all compute nodes.

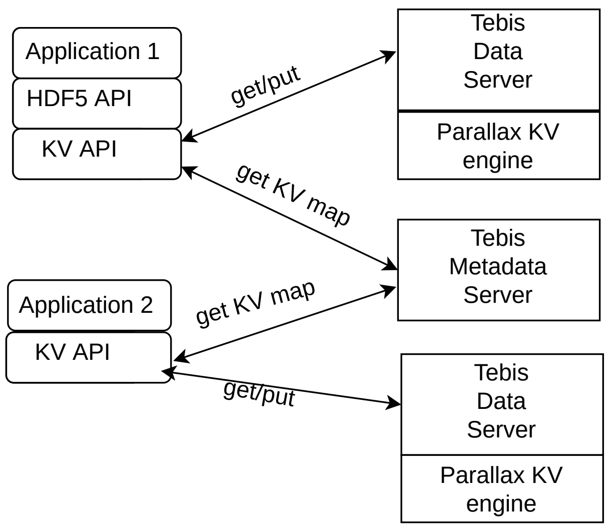


FIGURE 14: TEBIS DISTRIBUTED KV STORE ARCHITECTURE

Benefits

HPC applications, such as mesh solvers, that generate intermediate data that do not fit in memory can benefit from using the KV store service in the following way. The KV store enables applications to split their dataset into variable-size chunks to match the machine's capabilities without hurting performance. KV store can ingest data at a constant rate regardless of I/O sizes and b) can serve read I/O requests with a single I/O operation from the device.

Offered API

Applications can use the Tebis KV store either 1) through its native key-value API or 2) through HDF5 API. Tebis offers a Virtual Object Plugin interface for seamless integration of HPC applications that use HDF5.

Timeline

M18	First version of KV store available as open source: https://eupex.eu/sw-environment/sw-environment-storage-architecture
M36	Optimisations for reduced I/O latency.

5.2.2 Memory-Mapped I/O

Memory-mapped I/O (mmio) allows applications to transparently access data in storage devices via the page-fault mechanism, using processor load/store instructions. mmio has the potential to (a) eliminate modifications to applications for handling and processing large datasets by merely extending their heap over fast storage devices and (b) provide attractive abstractions for an application to control its I/O path over a unified data representation. Despite these advantages,

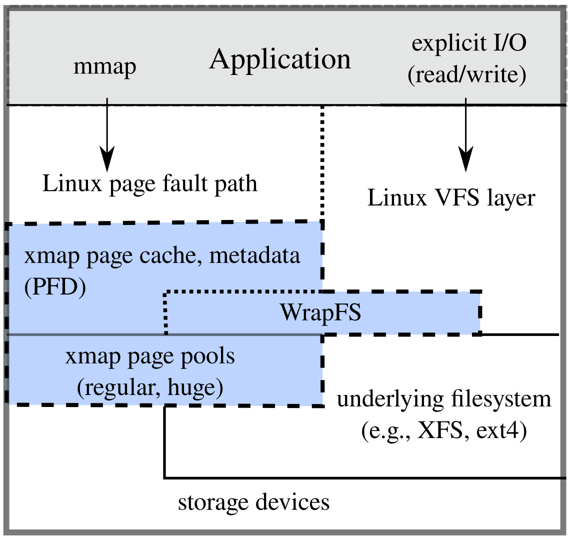

mmap
Application
explicit I/O (read/write)
Linux page fault path
Linux VFS layer
xmap page cache, metadata (PFD)
WrapFS
xmap page pools (regular, huge)
underlying filesystem (e.g., XFS, ext4)
storage devices

FIGURE 15: FASTMAP (XMAP) DESIGN OVERVIEW.

mmio has significant limitations that make it less attractive. In this work, we first analyse the current limitations of mmio. Then, we design xmap (Figure 15), an extension of FastMap for the Linux kernel, which addresses these limitations. The main contributions of xmap are support for transparent huge pages over block-based storage and asynchronous promotions. To our knowledge, xmap is the first system that provides this support for the Linux kernel. We evaluate xmap with a variety of graph processing workloads using Ligra, an in-memory graph processing framework, by transparently extending its heap over storage with no code modifications. Our results show that when processing graph datasets 6-8× larger than the available system DRAM, xmap outperforms Linux mmap by up to 3.5×, reduces total page faults by up to 265×, decreases CPU system time by up to 90%, and increases CPU user time by up to 250%.

Placement

FastMap/xmap is placed in the storage nodes and can support any storage services that require or can make use of memory-mapped I/O. xmap can also be used in compute nodes, if HPC applications make use of memory-mapped I/O over privately allocated (not shared) storage.

Benefits

FastMap/xmap eliminates the cost of hits in the I/O cache by using the virtual address translation layer in the CPU. As such, it reduces the cost of the I/O cache. In addition, xmap uses hugepages transparently, reducing both the cost of I/O and the use of the TLB for applications that use large amounts of I/O.

Offered API

FastMap/xmap uses the regular Linux kernel interface (mmap) for memory-mapped I/O.

Timeline

M18 First version of xmap is available as open source: https://eupex.eu/sw-environment/sw-environment-storage-architecture

M36 Policy optimisations for hugepages in xmap.

5.2.3 IO-SEA Ephemeral I/O Services

The Atos(Eviden) *Flash Accelerators* product is being extended in the IO-SEA project to become the run time environment of the data nodes. It provides ephemeral I/O services to workflows. Performance and activity metrics are collected to let users analyse the performance gains and the data movements savings. Health checking is also being added in order to make the solution robust against hardware failures.

The second piece of the IO-SEA solution is the Workflow Manager. It is in charge of user interaction through a set of commands. Ephemeral Services are made available to users within *sessions*. Workflows steps and I/O Ephemeral Services are described in a YAML file (the Workflow Description File) that is used to start a session (i. e., create the Ephemeral Services), then submit steps as needed (i. e., configure the selected compute nodes to set up access to the needed Ephemeral Services) until the session is stopped (i. e., terminate the Ephemeral Services and flush data to long term storage). The Workflow Manager processes the user commands, manages data nodes resources and orchestrates the deployment of Ephemeral Services.

Within EUPEX, the IO-SEA stack will be ported on the Rhea environment and the FORTH's KV Store will be integrated as an additional Ephemeral Service.

Timeline

M30 Compilation and preparation of the ARM A64FX environment

- Porting of the IO-SEA solution on ARM
- Work with WP3 use cases to create their workflow description files.
- integrate KV Store as an Ephemeral Service

M36 Running on Rhea nodes.

- Deployment of the IO-SEA Stack on the prototype.
- tests of the KV Store
- tests with WP3 use cases

5.3 Application Specific Interfaces

The ECMWF domain-specific object store for meteorology (FDB[2]) will be further developed to make best use of the offered storage services. The unique feature of FDB lies in its use of scientific semantic indexing, which abstracts the complexities of IO hardware technology, enabling application developers to manage data storage and access solely through scientific metadata. The primary objective is to support meteorological use-cases, such as the Climate and Weather Forecast application in WP3, to utilise the NVMe storage available on datanodes.

In order to achieve this, a 'remote' FDB will be installed directly on the datanodes. This will provide direct access to NVMe resources and function as a data storage server. Parallel applications executing on compute nodes will be able to read input and send output directly to this server, facilitating high-throughput, low-latency data storage.

Upgrades to the FDB will be implemented to enable seamless access to multiple storage tiers simultaneously, so that this FDB is able to tranparently route IO requests to, or move objects between, NVMe and the parallel file system.

Additionally, it is feasible to develop an object storage backend layer that will facilitate the storage of FDB objects in the KV stores from FORTH (see section 5.2.1). This possibility will be explored further.

Progress made in the IO-SEA project to develop a domain-agnostic scientific object store, known as DASI, will directly benefit from these advancements.[3] DASI, a metadata-driven data store inspired by and built on top of the FDB, can be used for non-meteorological applications as well.

Timeline

M36	FDB installed on datanodes with NVMe access set up for meteorological use-cases.
M48	Seamless, transparent data access to both NVMe and parallel file system via the FDB.

5.4 Tools and Acceleration Libraries

5.4.1 IO Instrumentation and the IO-SEA Recommendation System

IO-Instrumentation

The IO-SEA stack includes 2 user targeted tools to help them understanding the I/O aspects of their workflows and properly size their ephemeral I/O services: *IO-Instrumentation* and the *IO-SEA Recommendation System*. IO-Instrumentation is designed to collect the I/O activity of workflow steps and their usage of Ephemeral Services. It gather many statistics such as the I/O volumes, the time taken by each I/O operation as seen by the steps,... and create records for each 5 second timeslices of the run time in a condensed format. Metrics are gathered form compute

[2] https://github.com/ecmwf/fdb
[3] https://github.com/ecmwf-projects/dasi

nodes and stored in a central Mongo DB database. A graphical user interface is available to display those metrics on a per-job and per-workflow basis.

Applications are instrumented through an interception mechanism at the GLIBC level (LD_PRELOAD), so neither modifications nor recompilations of applications are necessary. However, it is possible to have conflicts with other middlewares that rely on the same interception mechanism.

Instrumentation is automatically enabled by the IO-SEA Workflow Manager when used. It is also possible to set it up defining a few environment variables. The IO-Instrumentation product is deployed on compute nodes for collecting metrics, and on one or more service nodes for the database and the Graphical User Interface.

IOSEA Recommendation System

The IO-SEA Recommendation System is a tool that analyses IO-Instrumentation records of a workflow and provides recommendation in term of data placement and configuration of the Ephemeral Services in order to optimise either in terms of performance or energy consumption. The recommendation are done through settings adjustments in the Workflow Description File. The system utilises a black box optimisation engine, a mathematical model of the application's I/O behaviour, a machine learning-based performance model, and a simulation engine to efficiently explore the parameter space and simulate application execution in a virtual HPC environment. Our approach selects the most suitable storage tier for each I/O phase, minimising the workflow's total duration and Burst Buffer usage.

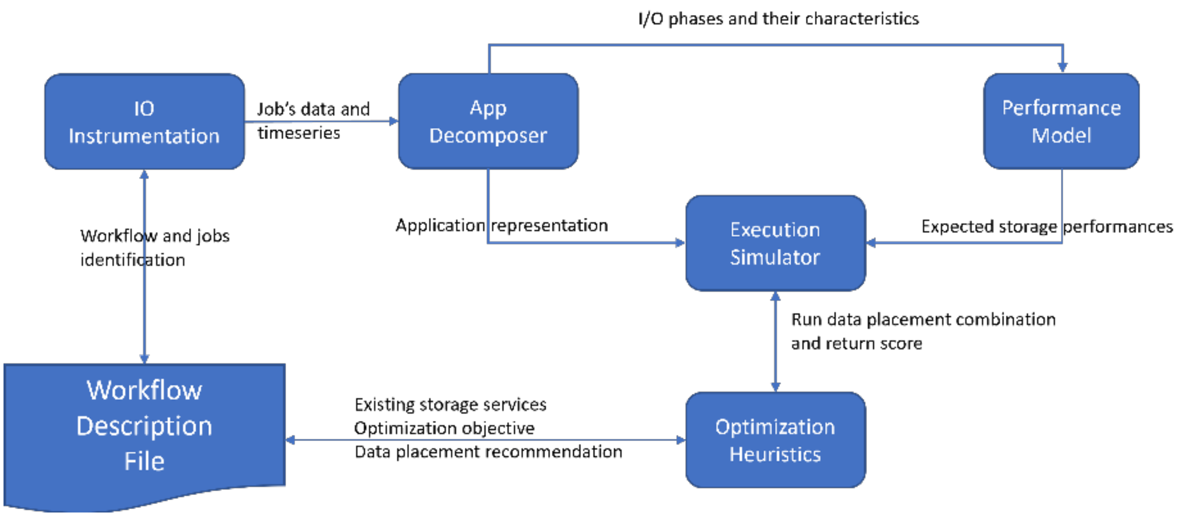

I/O phases and their characteristics
IO Instrumentation
Job's data and timeseries
App Decomposer
Performance Model
Workflow and jobs identification
Application representation
Execution Simulator
Expected storage performances
Run data placement combination and return score
Workflow Description File
Existing storage services
Optimization objective
Data placement recommendation
Optimization Heuristics

FIGURE 16: THE GENERAL ARCHITECTURE OF THE RECOMMENDATION SYSTEM.

In EUPEX, the recommendation system will be deployed and a performance model of the test infrastructure will be created. Then, it will be used with use cases to optimise their usage of the ephemeral services.

Timeline

M30 Compilation and tests on ARM A64FX

- Port IO-Instrumentation and the Recommendation System on Rhea
- Work with WP3 use cases to create their workflow description files.

M36 Running on Rhea nodes

- Experiment with use cases the usage of Ephemeral Services

5.4.2 ParaStation HealthChecker

The ParaStation HealthChecker will be used to ensure that at the start time of each new job the IO-SEA data nodes to be used by the job are in a defined health state and all services, system services as well as ephemeral services, are up and running as expected. The HealthChecker will be triggered by the ephemeral I/O services startup mechanism. Health checking tests suitable for the special environment on the IO-SEA data nodes will be installed and configured.

In EUPEX this setup and configuration will be deployed and further adjustments of the tests are made to the hardware of the prototype.

5.4.3 MAWA-HPC: Modular and Automated Workload Analysis for HPC Systems

Given the complexity of modern HPC systems, achieving theoretical peak performance depends on a myriad of parameters and system configurations. In order to optimise the system performance and efficiently use the underlying resources, various methods can be applied, including simulation, benchmarking, and monitoring. However, these methods and the tools are not compatible with each other and only consider a selection of performance factors such network, Input / Output (I/O), resource allocation, or parallel execution. Yet, each of these approaches generate knowledge that can be applied to similar problems or system configurations. To avoid that such knowledge is collected only for one-time purposes, and to also support other users, this knowledge must be easily accessible and available to the community.

The MAWA-HPC (Modular and Automated Workload Analysis for HPC Systems) [45] infrastructure aims to provide a generic workflow and tooling suite that can be applied to different use cases and workloads from different science domains. By integrating JUelich Benchmarking Environment (JUBE) for automation, established monitoring and profiling tools, node-level performance engineering tools, network benchmarks, and microbenchmarks for various parallel programming models (e. g., MPI and PGAS), MAWA-HPC enables multidimensional performance modelling. This makes established community tools available in a user-friendly manner and facilitates the identification and understanding of system and application level performance anomalies through the development of intuitive dashboards and hints. Through its modular design, the workflow is able to support various community profiling and monitoring tools, increasing the compatibility of each tool and covering new use cases. Figure 17 shows the high-level system design of MAWA-HPC and its current prototype implementation.

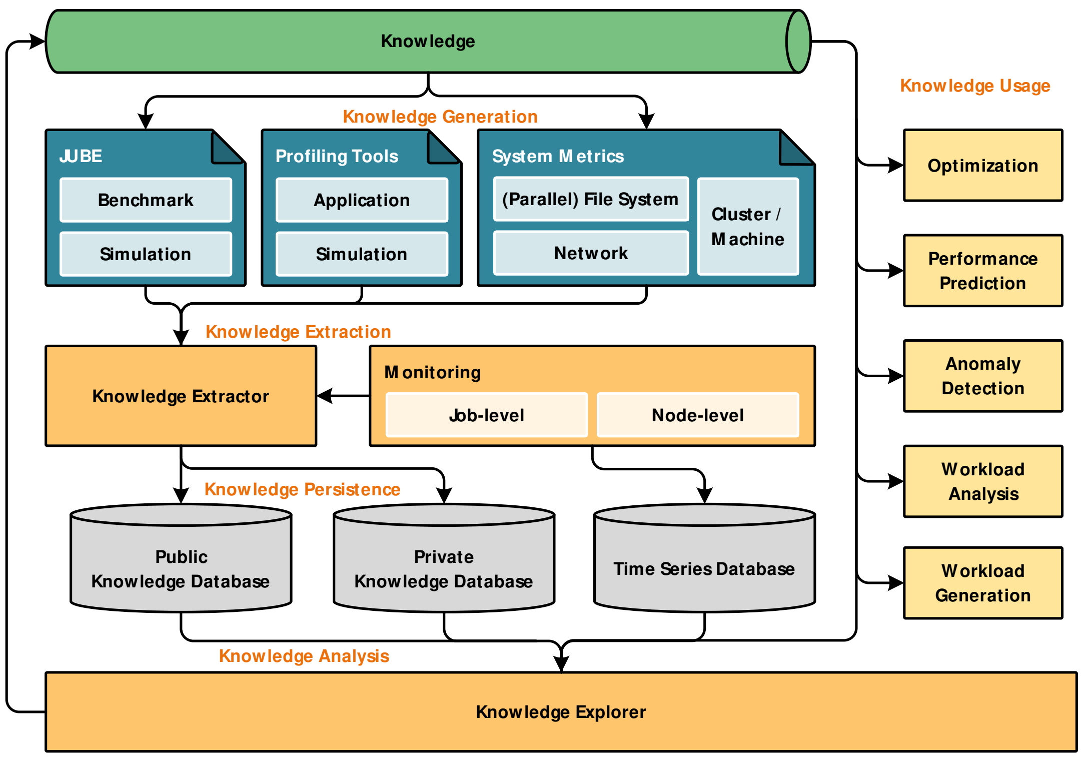


FIGURE 17: HIGH-LEVEL ARCHITECTURE OF THE MODULAR AND AUTOMATED WORKLOAD ANALYSIS FOR HPC SYSTEMS (MAWA-HPC) ANALYSIS CYCLE.

Developments/Extensions planned within EUPEX

MAWA-HPC will be adapted to the special characteristics of the EUPEX platform. In the first step, Goethe University Frankfurt plans to integrate the support of the community tools Score-P and Scalasca, which are ported and adapted to EUPEX by Forschungszentrum Jülich in Task 5.3. By extending MAWA-HPC with the support for additional monitoring and profiling tools, Goethe University intends to introduce a multi-dimensional Roofline model for performance characterisation [46, 47]. Including time as a third dimension, the Roofline model can provide insight into an application's performance over time, enabling the identification and understanding of I/O performance anomalies.

Subsequently, support for I/O load balancing (for both application data and metadata) based on the analysis of the I/O behaviour of workflows to distribute the workload evenly over all available storage targets will be integrated, especially for burst I/O where the shared nature of parallel file and storage systems causes load imbalance and resource contention over the end-to-end I/O paths resulting in severe performance degradation. To achieve this, Goethe University plans to integrate the end-to-end I/O load balancing framework Tarazu with MAWA-HPC.

Timeline

M30	Score-P and Scalasca integrated into the MAWA-HPC tooling infrastructure.
M36	Prototype integration of Tarazu with MAWA-HPC.

M48 MAWA-HPC ported and adapted to EUPEX pilot platform.

5.5 Hardware Dependencies on the Prototype

Discussions have been conducted with Work Package 4 to define the requirements of the EUPEX I/O stack. These discussions led to 2 specific requests:

- Local disks on Compute nodes, in order to run the tools of the in-memory tier
- Data Nodes equipped with NVMe storage and fast connections to the different compute modules, in order to run the Ephemeral I/O Services

6 Summary

This deliverable presents the software ecosystem of the EUPEX platform. This ecosystem covers all the technical areas critical to the operation and maintenance of an MSA system. These range from system management components forming the control plane, via the execution environment providing applications with a unified view on the available resources; through to tools for tracing, monitoring, and optimisation to ensure the efficient execution of applications and workflows on the Pilot system. In addition, the EUPEX software suite includes an I/O stack enabling the efficient access to hierarchical, multi-tier I/O subsystems. While presenting the individual components and how they fit into the software ecosystem, this deliverable also provides details on the timelines with respect to further developments and integration within EUPEX.

The EUPEX software ecosystem will be the result of integrating various components from different projects, companies, and research institutions across Europe to form a single, coherent, and mature software stack to operate Exascale-class MSA systems. Although the software stack will be designed with the EUPEX Pilot system in mind, it is not bound to the characteristics of this particular platform but will support MSA-based supercomputers in general.

Acronyms and Abbreviations

A

aarch64 64-bit extension of the ARM architecture family 10
ACCO *(Atos Collective Communications Optimizer)* 20
API *(Application Programming Interface)* 6, 49
ARM *(Advanced RISC Machines)* Family of Reduced Instruction Set Computing (RISC) architectures for computer processors, configured for various environments. Formerly standing for Advanced RISC Machine, or Acorn RISC Machine 12
Atos Bull SAS, France. 18

B

BHCO Bull Hybrid Communication Optimizer 20, 21
BMC *(Baseboard Management Controller)* A unit in a compute node enabling out-of-band management of the hardware. 11
BXI *(BullSequana eXascale Interconnect)* 17–19, 49

C

CEA *(Commissariat à l'énergie atomique et aux énergies alternatives)* Commissariat à l'énergie atomique et aux énergies alternatives, France 8, 9, 29
CPU *(Central Processing Unit)* 9, 19, 21
CWL *(Common Workflow Language)* An open standard for describing how to run command line tools and connect them to create workflows 22

D

DEEP-SEA *(DEEP – Software for Exascale Architectures)* 19

E

EAP *(EUPEX Alfa Pilot platform)* Already existing system with similar characteristics as the EUPEX system. This will serve as Software Development Vehicle (SDV) for application developers enabling the adapation of theirs codes. 11, 17, 19–21, 29
EUPEX *(European Pilot for eXascale)* 6–11, 14–26, 28–32, 34, 35, 41, 43–48, 50

F

FORTH *(Foundation for Research and Technology – Hellas)* Foundation for Research and Technology – Hellas, Greece 8

G

GASPI *(Global Address Space Programming Interface)* 19
GPI-2 Global address space Programming Interface version 2.0: an open source implementation of the GASPI standard under GPL v3 19
GPU *(Graphics Processing Unit)* 9, 19

H

HAN *(Hierarchical AutotuNed)* A collective communication framework 18, 20
HPC *(High Performance Computing)* 8, 11, 12, 16, 22–24, 44, 49

I

I/O *(Input / Output)* 44, 45
I/O *(Input / Output)* 6, 7, 47

Bibliography

[1] E. Suarez, N. Eicker, and T. Lippert. "Modular Supercomputing Architecture: from Idea to Production; 3rd". In: *Contemporary High Performance Computing: From Petascale toward Exascale, Volume 3*. Vol. 3. 2019, pp. 223–251.

[2] S. Neuwirth. "Modular Supercomputing and its Role in Europe's Exascale Computing Strategy". In: *Proceedings of The 39th International Symposium on Lattice Field Theory — PoS(LATTICE2022)*. Vol. 430. 2023, p. 245. DOI: 10.22323/1.430.0245.

[3] *Ocean repository*. URL: https://ocean.eupex.eu/ (visited on 04/25/2023).

[4] *Irene webiste*. URL: https://www-hpc.cea.fr/en/Joliot-Curie.html (visited on 04/25/2023).

[5] W. Wang et al. "Exploring the Feasibility of Fully Homomorphic Encryption". In: *IEEE Transactions on Computers* 64 (Mar. 2015), pp. 698–706. DOI: 10.1109/TC.2013.154.

[6] A. Viand, P. Jattke, and A. Hithnawi. "SoK: Fully Homomorphic Encryption Compilers". In: May 2021. DOI: 10.1109/SP40001.2021.00068.

[7] M. Schneider et al. "Sok: Hardware-supported trusted execution environments". In: *arXiv preprint arXiv:2205.12742* (2022).

[8] *ARM Trust Zone*. URL: https://www.arm.com/technologies/trustzone-for-cortex-a/ (visited on 05/16/2023).

[9] *AMD SEV*. URL: https://developer.amd.com/sev/ (visited on 05/16/2023).

[10] *Intel SGX*. URL: https://www.intel.com/content/www/us/en/architecture-and-technology/software-guard-extensions.html (visited on 05/16/2023).

[11] *OP-TEE*. URL: https://www.op-tee.org (visited on 05/16/2023).

[12] S. Pickartz. "Virtualization as an enabler for dynamic resource allocation in HPC". PhD thesis. 2019. DOI: 10.18154/RWTH-2019-02208.

[13] D. Grünewald and C. Simmendinger. "The GASPI API specification and its implementation GPI 2.0". In: *7th International Conference on PGAS Programming Models*. Vol. 243. 2013.

[14] S. Neuwirth. "Toward a Comprehensive Benchmark Suite for Evaluating GASPI in HPC Environments". In: *2021 IEEE International Conference on Cluster Computing (CLUSTER)*. 2021, pp. 827–828. DOI: 10.1109/Cluster48925.2021.00082.

[15] S. Cherubin and G. Agosta. "libVersioningCompiler: An easy-to-use library for dynamic generation and invocation of multiple code versions". en. In: *SoftwareX* 7 (Jan. 2018), pp. 95–100.

[16] I. Colonnelli et al. "StreamFlow: cross-breeding cloud with HPC". In: *IEEE Transactions on Emerging Topics in Computing* 9.4 (2021), pp. 1723–1737. DOI: 10.1109/TETC.2020.3019202.

[17] F. Broquedis et al. "hwloc: a Generic Framework for Managing Hardware Affinities in HPC Applications". In: *Proceedings of the 18th Euromicro International Conference on Parallel, Distributed and Network-Based Processing (PDP2010)*. Pisa, Italia: IEEE Computer Society Press, Feb. 2010, pp. 180–186. DOI: 10.1109/PDP.2010.67. URL: http://hal.inria.fr/inria-00429889.

[18] *Portable Hardware Locality (hwloc)*. URL: https://www.open-mpi.org/projects/hwloc/ (visited on 04/28/2023).

[19] B. Goglin. “Exposing the Locality of Heterogeneous Memory Architectures to HPC Applications”. In: *The Second International Symposium on Memory Systems Proceedings (MEMSYS16)*. Washington, DC: ACM, Oct. 2016, pp. 30–39. DOI: 10.1145/2989081.2989115. URL: https://hal.inria.fr/hal-01330194.

[20] B. Goglin and A. R. Proa~no. “Using Performance Attributes for Managing Heterogeneous Memory in HPC Applications”. In: *The 23rd IEEE International Workshop on Parallel and Distributed Scientific and Engineering Computing (PDSEC 2022), held in conjunction with IPDPS 2022*. Lyon, France: IEEE, May 2022. DOI: 10.1109/IPDPSW55747.2022.00145. URL: http://hal.inria.fr/hal-03599360.

[21] O. Vysocky et al. “Evaluation of the HPC Applications Dynamic Behavior in Terms of Energy Consumption”. In: *Proceedings of the Fifth International Conference on Parallel, Distributed, Grid and Cloud Computing for Engineering*. 2017, pp. 1–19. DOI: 10.4203/ccp.111.3.

[22] J. Schuchart et al. “The READEX formalism for automatic tuning for energy efficiency”. In: *Computing* 99.8 (Aug. 2017), pp. 727–745. ISSN: 1436-5057. DOI: 10.1007/s00607-016-0532-7.

[23] O. Vysocky et al. “Analysis and Visualization of the Dynamic Behavior of HPC Applications”. In: *High Performance Computing in Science and Engineering*. Ed. by T. Kozubek et al. Cham: Springer International Publishing, 2021, pp. 135–149. ISBN: 978-3-030-67077-1. DOI: 10.1007/978-3-030-67077-1_8.

[24] Y. Kodama, M. Kondo, and M. Sato. “Evaluation of SPEC CPU and SPEC OMP on the A64FX”. In: *2021 IEEE International Conference on Cluster Computing (CLUSTER)*. Los Alamitos, CA, USA: IEEE Computer Society, Sept. 2021, pp. 553–561. DOI: 10.1109/Cluster48925.2021.00088.

[25] *Arm System Control and Management Interface: Platform Design Document*. Tech. rep. 3.2. ARM, 2022.

[26] A. Knüpfer et al. “Score-P: A Joint Performance Measurement Run-Time Infrastructure for Periscope,Scalasca, TAU, and Vampir”. In: *Tools for High Performance Computing 2011*. Ed. by H. Brunst et al. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 79–91. DOI: 10.1007/978-3-642-31476-6_7.

[27] *Score-P website*. URL: https://score-p.org/ (visited on 05/09/2023).

[28] A. Knüpfer et al. “The Vampir Performance Analysis Tool-Set”. In: *Tools for High Performance Computing*. Ed. by M. Resch et al. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, pp. 139–155. DOI: 10.1007/978-3-540-68564-7_9.

[29] S. S. Shende and A. D. Malony. “The Tau Parallel Performance System”. In: *The International Journal of High Performance Computing Applications* 20.2 (2006), pp. 287–311. DOI: 10.1177/1094342006064482.

[30] A. Calotoiu et al. “Using Automated Performance Modeling to Find Scalability Bugs in Complex Codes”. In: *Proc. of the ACM/IEEE Conference on Supercomputing (SC13), Denver, CO, USA*. ACM, Nov. 2013, pp. 1–12. ISBN: 978-1-4503-2378-9. DOI: 10.1145/2503210.2503277.

[31] D. Eschweiler et al. “Open Trace Format 2 – The Next Generation of Scalable Trace Formats and Support Libraries”. In: *Advances in Parallel Computing (Proc. of the Intl. Conference on Parallel Computing, ParCo)* 22 (2012), pp. 481–490. DOI: 10.3233/978-1-61499-041-3-481.

[32] M. Geimer et al. “Further Improving the Scalability of the Scalasca Toolset”. In: *Applied Parallel and Scientific Computing*. Ed. by K. Jónasson. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 463–473. DOI: 10.1007/978-3-642-28145-7_45.

[33] D. Terpstra et al. “Collecting Performance Data with PAPI-C”. In: *Tools for High Performance Computing 2009*. Ed. by M. S. Müller et al. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 157–173. DOI: doi.org/10.1007/978-3-642-11261-4_11.

[34] S.-P. developer community. *Scalable performance measurement infrastructure for parallel codes (Score-P)*. Version 8.1. Apr. 2023. DOI: 10.5281/zenodo.7817192. URL: https://doi.org/10.5281/zenodo.7817192.

[35] M. Geimer et al. “A scalable tool architecture for diagnosing wait states in massively parallel applications”. In: *Parallel Computing* 35.7 (July 2009), pp. 375–388. ISSN: 0167-8191. DOI: https://doi.org/10.1016/j.parco.2009.02.003.

[36] I. Zhukov et al. “Scalasca v2: Back to the Future”. In: *Tools for High Performance Computing 2014*. Ed. by C. Niethammer et al. Springer International Publishing, 2015, pp. 1–24. DOI: 10.1007/978-3-319-16012-2_1.

[37] *Scalasca website*. URL: https://scalasca.org/ (visited on 05/09/2023).

[38] D. Böhme et al. “Identifying the Root Causes of Wait States in Large-Scale Parallel Applications”. In: *ACM Trans. Parallel Comput.* 3.2 (July 2016). ISSN: 2329-4949. DOI: 10.1145/2934661.

[39] D. Böhme et al. “Scalable Critical-Path Based Performance Analysis”. In: *2012 IEEE 26th International Parallel and Distributed Processing Symposium*. May 2012, pp. 1330–1340. DOI: 10.1109/IPDPS.2012.120.

[40] S. developer community. *Toolset for scalable performance analysis of large-scale parallel applications (Scalasca)*. Version 2.6.1. Dec. 2022. DOI: 10.5281/zenodo.7440854. URL: https://doi.org/10.5281/zenodo.7440854.

[41] P. Saviankou et al. “Cube v4: From Performance Report Explorer to Performance Analysis Tool”. In: *Procedia Computer Science* 51 (June 2015), pp. 1343–1352. DOI: 10.1016/j.procs.2015.05.320.

[42] P. Saviankou, A. Visser, and C. developer community. *CubeGUI: Graphical explorer*. Version 4.8.1. Mar. 2023. DOI: 10.5281/zenodo.7737411. URL: https://doi.org/10.5281/zenodo.7737411.

[43] G. Xanthakis et al. “Parallax: Hybrid Key-Value Placement in LSM-Based Key-Value Stores”. In: *Proceedings of the ACM Symposium on Cloud Computing*. SoCC ’21. Seattle, WA, USA: Association for Computing Machinery, 2021, pp. 305–318. ISBN: 9781450386388. DOI: 10.1145/3472883.3487012. URL: https://doi.org/10.1145/3472883.3487012.

[44] M. Vardoulakis et al. “Tebis: Index Shipping for Efficient Replication in LSM Key-Value Stores”. In: *Proceedings of the Seventeenth European Conference on Computer Systems*. EuroSys ’22. Rennes, France: Association for Computing Machinery, 2022, pp. 85–98. ISBN: 9781450391627. DOI: 10.1145/3492321.3519572. URL: https://doi.org/10.1145/3492321.3519572.

[45] Z. Zhu, N. Bartelheimer, and S. Neuwirth. *MAWA-HPC: Modular and Automated Workload Analysis for HPC Systems*. Research Poster, ISC High Performance 2023 (ISC23). 2023.

[46] N. Bartelheimer, Z. Zhu, and S. Neuwirth. “Toward a Modular Workflow for Network Performance Characterization”. In: *25th Workshop on Advances in Parallel and Distributed Computational Models (APDCM’23), held in conjunction with IPDPS’23*. 2023. DOI: 10.1109/IPDPSW59300.2023.00062.

[47] Z. Zhu, N. Bartelheimer, and S. Neuwirth. “An Empirical Roofline Model for Extreme-Scale I/O Workload Analysis”. In: *4th Workshop on on Extreme-Scale Storage and Analysis (ESSA’23), held in conjunction with IPDPS’23*. 2023. DOI: 10.1109/IPDPSW59300.2023.00106.