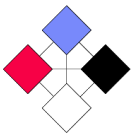
D1.6

Final Project Report



DAPHNE

Integrated Data Analysis Pipelines for Large-Scale Data Management, HPC, and Machine Learning

Version 1.4

PUBLIC

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No. 957407.

Document Description

In D1.6, the DAPHNE project team describe the progress made until project month 48 and the work done in project year 4 (M37/Dec 2023 – M48/Nov 2024). This report presents an overview of the type and purpose of the document, its revision history, the strategic objectives of DAPHNE project and the work carried out in the last project period to reach these objectives. Then, a more detailed description concerning work done in the last year across all work packages (WPs) is provided.

D1.6 Final Project Report			
WP1 – Project Management			
Type of document	R	Version	4.0
Dissemination level	PU	Project month	48
Lead partner	KNOW		
Author(s)	all		
Reviewer(s)	Ilin Tolovski (HPI)		

Revision History

Version	Revisions and Comments	Author / Reviewer
V1.0	Initial draft and structure	Eva Paulusberger (KNOW)
V1.1	Text sections by responsible partners	Mark Dokter (KNOW), Eva Paulusberger (KNOW), Patrick Damme (TUB), Matthias Boehm (TUB), Philipp Ortner (TUB), Nils Strassenburg (HPI), Ilin Tolovski (HPI), Tilmann Rabl (HPI), Andreas Laber (IFAT), Philippe Bonnet (ITU), Benjamin Steinwender (KAI), Dirk Habich (TUD), Florina M. Ciorba (UNIBAS),

		Jonas Korndorfer (UNIBAS), Dimitrios Tsoumakos (ICCS)
V1.2	Semantic and syntactic adaptations	Eva Paulusberger (KNOW)
V1.3	Review 1	Ilin Tolovski (HPI)
V1.4	Finalization	Eva Paulusberger (KNOW)

Abbreviations

Abbreviaton	Full term
ABS	Access and Benefit Sharing
D	Deliverable
DM	Data Management
T	Task
HPC	High-Performance Computing
KNOW	Know Center Research GmbH
ML	Machine Learning
IPR	Intellectual Property Rights
ABS	Access and Benefit Sharing
WP	Work Package

1 Introduction and Purpose of this Document

In D1.6, the DAPHNE project team describes the progress made from project month 37 to project month 48, respectively the work done in the last project year (M37/Dec 2023 – M48/Nov 2024).

First, this report refers to the structure and purpose of the document. Second, D1.6 outlines the main objectives in DAPHNE and what DAPHNE consortium has done to reach those targets. In the third section of this final report an overview of achievements across the work packages 1 to 10 is presented, particularly addressing the work in the last project year 2024. The purpose of this document is therefore to provide an overview of DAPHNE project until M48, with an emphasis on updates of the last project year.

2 Strategic Objectives

This section shows the strategic objectives and the work of DAPHNE consortium towards these objectives in the 4th project year.

2.1. Objective 1 System Architecture, APIs and DSL (WP2-4)

Objective 1 System Architecture, APIs and DSL: *Improve the productivity for developing integrated data analysis pipelines via appropriate APIs and a domain-specific language, an overall system architecture for seamless integration with existing data processing frameworks, HPC libraries, and ML systems. A major goal is an open, extensible reference implementation of the necessary compiler and runtime infrastructure to simplify the integration of current and future state-of-the-art methods.*

While the first years of the DAPHNE project have been used to build up the system infrastructure, the progress towards the end has shifted from creation to refinement-oriented. This has already been visible in the last report and has been continued in this final project report. While the runtime system has seen many new kernels to enrich the functionality or

improve the performance, there a lot of bug fixes and convenience features that were introduced for developers and users alike. For the general system architecture, the first version of the extensibility catalogue has been mainlined. The DAPHNE compiler gained more code generation capabilities, optimization and analysis passes. The API was extended continuously in our DaphneLib python bindings and our DaphneDSL has seen improvements for productivity like better support for indexing into matrix structures and higher level built-in functions, which in turn, sparked the development of internal features like the list data type, fostering the implementation of algorithms like the decision trees.

2.2 Objective 2 Hierarchical Scheduling and Task Planning (WP5-WP7)

Objective 2 Hierarchical Scheduling and Task Planning: *Improve the utilization of existing computing clusters, multiple heterogeneous hardware devices, and capabilities of modern storage and memory technologies through improved scheduling as well as static (compile time) task planning. In this context, we also aim to automatically leverage interesting data characteristics such as sorting order, degree of redundancy, and matrix/tensor sparsity.*

Besides refinement, the progress in the final project year towards reaching the goals of the second strategic objective was also marked by the development of larger features in separate branches to not interfere with general development in the main branch while they are not mature enough for merging. Relevant changes in this regard were the integration of distributed I/O through HDFS and Lustre support or improved memory management to support sparse data on GPU, support for the io_uring subsystem, higher dimensional tensor data and ongoing work in the computational storage and FPGA kernels. Work on hierarchical scheduling and NU-MA aware data placement was carried out to cater to the objective's task planning aspects, leading to performance improvements by countering load imbalance. Many of the techniques for this objective are based on features that have been introduced or improved in the DAPHNE compiler that analyzes workloads, input data and intermediates. Based on these outputs, decisions can be made on which methods are best applied to the problem at hand.

2.3 Objective 3 Use Case Studies and Benchmarking (WP8-WP9)

Objective 3 Use Cases and Benchmarking: *The technological results will be evaluated on a variety of real-world use cases and datasets as well as a new benchmark developed as part of the DAPHNE project. We aim to improve the accuracy and runtime of real-world use cases combining data management, machine learning, and HPC – this exploratory analysis serves as a qualitative study on productivity improvements (Objective 1). The variety of real-world use cases will further be generalized to a benchmark for integrated data analysis pipelines quantifying the progress compared to state-of-the-art (Objective 2).*

The progress towards completion of the use case implementations and our benchmarking efforts has been naturally increasing towards the end of the project as DAPHNE becomes more and more mature and therefore usable in real world application as well as benchmarking scenarios. In collaboration with the technical work packages (WPs 3-7), the work packages for use cases (WP8) and benchmarking (WP9) were not only able to reach their final stages, but also gave valuable feedback where problems still needed to be solved and improvements to be implemented. Through our successful inter-WP collaboration we were able to have most use cases produce their results with the UMLAUT benchmarking suite that has been implemented alongside the DAPHNE development in a separate code repository.

3 Status and Progress Update of All Work Packages

This section provides an overview of the progress made in the last project year throughout the work packages.

3.1 WP1 Project Management (led by KNOW) [M1-M48]

WP1 Project Management provides a high-quality work environment for research activities to thrive, to coordinate across WPs, to keep track of reporting and deliverables and to improve communication throughout the consortium, the European Commission and beyond. In this

section WP1 Lead KNOW reports about the objectives of WP1 and the work towards those objectives in the last project year.

Regarding the main objectives of this WP (1) to act as the communication interface with the European Commission, DAPHNE project management has sought to share information effectively via the EU portal, reporting on all relevant continuous reporting items (deliverables, milestones, risks, publications, dissemination and communication, patents/IPR, innovation, open data, gender, ABS regulation) and communicate with the programme officer (PO) on relevant topics such as organizing meetings (i.e. organization of General Assembly and Review meetings), aligning on reporting (i.e. workflow for Periodic Report), asking for changes to the Grant Agreement and support (Budget Shift Agreement between KNOW and TUB) or reaching out for dissemination purposes.

Concerning WP1 objective (2) to establish means of effective communication and collaboration within the consortium, DAPHNE project management has reported on these means in D1.1 Project and Risk Management Plan [1]. In line with this plan, we have kept our basic 4-level project structure, used the, in D1.1 [1] described, mailing lists and tools, e.g., Daphne cloud and GitLab, as well as file storages for communication and collaboration purposes. We have migrated the development from a private GitLab instance to GitHub and maintained our DAPHNE registration procedure to ensure a smooth transition into (and out of) the project communication platforms and channels. Complementary to these channels and platforms we use regular WP-specific meetings, bilateral meetings, as well as monthly All-hands meetings across the entire consortium to ensure accurate and high-quality communication.

Resulting in WP1 objective (3) the organization of calls and meetings of the consortium, we hold our All-hands meetings monthly. In these consortium meetings, we discuss administrative and team updates, WP/technical updates, reporting and deliverable tracking such as news on for example publications and conferences; every month there are updates from all WP leads. The meeting serves the purpose of bringing together all consortium members, providing relevant information in a structured way, asking for support and alignment if required and giving everyone the chance to clarify questions that are preferred to be discussed orally within the whole consortium.

In addition to these regular general meetings, project consortium has met for the general assembly meeting once a year. The purpose and general outline of this meeting format have

been elaborated in various documents such as the Proposal, the Grant Agreement or the Project Plan (D1.1) [1]. Updates are that now, as we are at the end of the final project year, multiple demo presentations complement the WPs and UCs presentations and give more variety to the meeting. Moreover, in this last project year we continued to meet personally for the General Assembly Meeting. We experienced a boost in motivation based on this personal encounter.

WP1 Project Management reinforces the communication structure devised in D1.1 Project and risk management plan [1] and reminds the entire consortium to be aware of the objectives and the related deliverables. Specific internal project management tracking tools are the deliverable and reviewer tracker, the exploitation tracker and the financial tracking; their results are filtered and – depending on confidentiality-level - reported further within the EU portal and/or our project website [2] as well as to communication experts. Moreover, this objective seeks to ensure strategic realignment in case of unforeseen circumstances. Strategic realignment in the last project year was necessary in terms of the budget shift agreement between TUB and KNOW and the involvement of the legal departments and finance departments.

Objective (5) addresses the coordination and quality assurance of reporting efforts. The submitted reports and deliverables that were submitted until the end of the first project period have been accepted. With the constructive feedback received, we have started to include more reporting details, complementing our cross-referencing, and improve our prototype documentation efforts in the second project period and particularly in this last project year. Moreover, we should highlight that all consortium partners are collaborating on the DAPHNE system and that this intense collaboration is the essence of our project. Financial controlling, budget and effort reporting have been carried out, depicting deviations from the original budget plan, and giving the consortium partners the chance to compensate for these deviations in the upcoming project periods. The current DAPHNE website https://DAPHNE-eu.eu [2] shows those reports that are available to the public in the section *Publications*.

The central project management endeavors to maintain a general project overview across all WPs and the budget, to relate the actual work being done to the original project plan and to ensure effective communication have been carried out in this third project year - with many lessons learned, such as the importance of clear and motivating communication, including the setting of boundaries, decent conflict management skills, a professional team, as well as high

responsiveness. Moreover, we found that for project management to thrive administrative/organizational and classic project management, communication, as well as technical skills need to be comprised in the consortium. These complementing skills result in a focus on the right priorities.

Eventually, a lesson learned is that as manager you need to observe and reflect on the project, the stakeholders/organizations, and the processes, and gain an overview, re-focusing on the higher European objectives, which are well thought-through, and set priorities. Here, patience and acting ethical, helpful, and constructive are key. In the last project year, it is important to demonstrate that we have met the promises made in the Grant Agreement, our project plan, and have made progress accordingly.

3.2 WP2 System Architecture (KNOW) [M1-M21]

The open system architecture that fosters extensibility and caters to the needs of data analysis and data processing in the fields of machine learning (ML) and high-performance computing (HPC), which was documented in deliverable D2.1 [3]. In addition, D2.2 [4] refers to the refined overall architecture and key design decisions of the DAPHNE system infrastructure as an open and extensible system for IDA pipelines, comprising query processing, ML, and HPC. WP 2 System Architecture was active until M21. The reporting period for this report D1.6 does therefore not cover WP2 activities.

3.3 WP3 Abstractions and Compilation (TUB) [M1-M48]

We continued the work on DAPHNE's DSL abstractions, mainly to facilitate user productivity, as well as the DAPHNE compiler, mainly to achieve extensibility and system efficiency. We summarize our key contributions in project year 4 below.

DSL Abstractions. We have already described the design of DAPHNE's language abstractions in detail in deliverable D3.1 [5] and implemented most of this design in project years 1-3. Thus, our contributions in project year 4 mainly serve to further improve the existing implementation

and to ensure smoother and more productive use of the system. To that end, our main contributions in project year 4 are:

(1) We significantly improved the *error handling* capabilities of DAPHNE. Errors can occur due to invalid user inputs (script errors or invalid data) and due to system failures (especially the hardware like accelerators or networking). We improved error handling in DAPHNE at two levels. First, at the *system-level*, DAPHNE already adopted a fail-fast behavior, i.e., errors are detected (e.g., by means of the validation of inputs) and the DAPHNE program execution is terminated gracefully. However, as our use-case partners increasingly use the DAPHNE system and write increasingly complex DaphneDSL/DaphneLib scripts, the need for more helpful error messages arose. Thus, we ensured that error messages presented to users are consistent and actionable. As a result, all error messages now include the responsible DaphneDSL source code location (file, line, column) as well as details on the error and the component it originated from. To this end, we make use of (a) MLIR's built-in location tracking features and (b) a new DAPHNE system component at the boundary of the compiler and the runtime (WP4) that associates each kernel call to a DaphneDSL location. Overall, these improved error messages simplify writing and debugging complex DaphneDSL scripts significantly and, therefore, facilitate user productivity. Second, at the *user script-level*, we added a new `stop()` built-in function to DaphneDSL. This function can be employed by DaphneDSL users to terminate the script execution, if their own application-specific error conditions are violated. For instance, a user's custom training algorithm may require the input data to be encoded in a certain way, such as categorical features recoded as non-negative integers.

(2) We introduced a new *list data type* to DaphneDSL. DaphneDSL lists are containers that can store an arbitrary number of matrices of homogeneous type, but heterogeneous shape. Lists can be accessed through operations like `append()` and `remove()` and are useful for maintaining the state of certain iterative data analysis algorithms, such as the training algorithm of decision trees, effectively (from a user's point of view) and efficiently (from the system's point of view). We already mentioned the idea of a list data type in deliverable D3.1 [5].

Besides these larger contributions to DAPHNE's DSL abstraction, we also added *numerous smaller features*, especially in close coordination with our use-case partners from WP8. These additional features include: (a) improvements of the syntax of DaphneDSL and DaphneLib (e.g., support for element expressions inside matrix literals, left/right indexing in DaphneLib through

Python's `[]`-operator), (b) additional built-in functions (e.g., `isNan()`, `typeOf()`, `stop()`), and (c) additional reusable high-level data science primitives implemented in DaphneDSL (e.g., PageRank, k-Means, multinomial logistic regression). These smaller additions also facilitate the productivity of DAPHNE users.

Compilation. We have already described the DAPHNE compiler design in deliverables D3.1 [5] and D3.4 [9] and provided initial prototypes in deliverables D3.2 [6] and D3.3 [7]. Based upon this work in project years 1-3, we further improved the DAPHNE compiler in project year 4. These improvements are described in detail in deliverable D3.5 [8]. In the following, we only summarize the most important contributions, which are:

(1) We implemented the *kernel extension catalog* described in deliverable D3.4 [9] as a new component of the DAPHNE compiler. This catalog stores essential information about all available pre-compiled kernels, such that the DAPHNE compiler can lower DaphneIR operations to suitable kernels. Internally, we refactored the DAPHNE compiler to rely only on the extension catalog for this lowering step, even for built-in kernels. At the same time, expert users can now extend DAPHNE by custom kernels following the three-step approach illustrated in the figure below by (1) implementing their custom kernels against a clearly defined interface in a stand-alone code base and compiling it as a shared library, (2) registering their extension with DAPHNE's extension catalog at run-time, i.e., without the need to re-build DAPHNE, and (3) using their extension kernels, which can happen either automatically based on the argument and result types of an operation or manually through optional kernel hints in DaphneDSL, which are respected by the DAPHNE compiler. Besides kernel extensibility, we also started the work on extensibility with respect to new data types and value types. For instance, we refactored decisive parts of the code base to support non-numeric value types. Based on this work, we added two different string representations as new value types to DAPHNE. In total, extensibility is at the heart of DAPHNE and simplifies embracing specialization in terms of new hardware devices, data and value types, and algorithms.

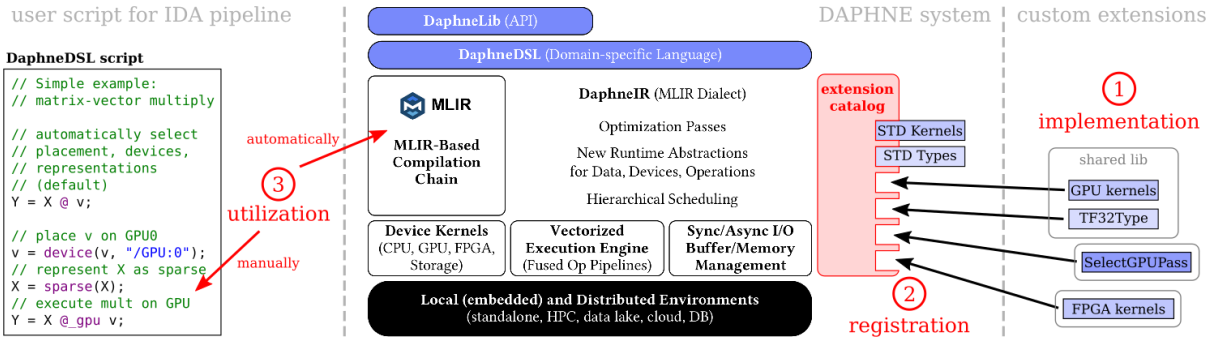


Figure 1: Kernel extension catalog

(2) We further extended the *MLIR-based CPU code generation backend*. On the one hand, we added code generation support for additional operations, such as matrix multiplication as well as full, row-wise, and column-wise aggregation. On the other hand, we explored the use of existing MLIR dialects and features for the code generation in DAPHNE. Generating low-level code for DaphneIR operations, rather than lowering them to calls to pre-compiled C++ kernels, can improve the runtime performance of a single operation, e.g., by exploiting DaphneDSL compile-time knowledge that is not available during C++ kernel compilation, and of a chain of operations, e.g., through operator fusion.

(3) We started to enhance DAPHNE's existing *operator fusion* capabilities with additional options for the data partitioning. As described in deliverable D2.2 [4], DAPHNE fuses operators into pipelines. These pipelines are executed by DAPHNE's vectorized execution engine, which pushes partitions of the input data through the pipeline in parallel. So far, operator fusion assumed that the data must be partitioned along the row axis. However, given that matrices have two dimensions, some operations may also support or even prefer partitioning along the column axis. We started implementing alternative operator fusion algorithms that make such options explicit to the DAPHNE compiler. The goal is to utilize more of the fusion potential in a DaphneDSL script and, thereby, to create longer fused pipelines for increased runtime performance. In that context, the choice of the physical representation of the pipeline's intermediate results is important, e.g., a sparse matrix represented in CSR (compressed sparse row) can be more efficiently accessed by row, while a sparse matrix represented in CSC (compressed sparse column) can be more efficiently accessed by column, even though both representations have a similar physical size in many cases.

Besides these three larger contributions that primarily concern the DAPHNE compiler itself, most of our work on the DSL abstractions mentioned above (e.g., error handling, list data type)

also required substantial additions to the DAPHNE compiler. Furthermore, we made *various smaller improvements* to the compiler, including: (a) a significant extension of the compiler's sparsity estimation support by adding naive meta data estimators for most DaphneIR operations and (b) improvements to the inter-procedural analyses (e.g., removal of unused functions, removal of (near) duplicate functions after specialization of untyped functions and constant propagation).

3.4 WP4 DSL Runtime and Integration (ICCS) [M1-M48]

We outline the updates and enhancements in the design and implementation of the DAPHNE Runtime system (work within WP4, detailed in Deliverable D4.4) in the final project year. We can categorize progress made in this period along the following axes: a) Daphne Runtime integration with File Systems, b) NUMA-aware data placement and c) Filetype support.

HDFS Integration

The integration of DAPHNE with the Hadoop Distributed File System (HDFS) provides significant benefits in big data and machine learning contexts by leveraging HDFS's scalability, ecosystem compatibility, data locality, fault tolerance, and high throughput. HDFS is designed for efficient storage and processing of large datasets, which makes it ideal for big data applications. It seamlessly integrates with tools like Hadoop, Spark, and Hive, creating a stable environment for machine learning applications that require access to diverse big data tools. Data locality within HDFS further enhances efficiency by reducing network latency through computations near the data's storage location. HDFS's fault tolerance is achieved through data replication, ensuring high availability and stability even in cases of hardware failure. Additionally, HDFS supports high throughput access, making it well-suited for managing large-scale data in big data and machine learning applications.

HDFS operates using a coordinator-worker architecture, with a central NameNode managing metadata and DataNodes storing data blocks. Files are split into 128 MB blocks by default, which are distributed across DataNodes to facilitate parallel processing and efficient load management. To maintain fault tolerance, each data block is replicated across several

DataNodes. This setup allows for quick access to data and easy recovery in the event of node failures.

The C++ API, facilitated by the libhdfs3 library, provides efficient access to HDFS from C/C++ applications without relying on Java dependencies. Libhdfs3 offers asynchronous I/O, allowing non-blocking data operations and improved interactivity in high-performance applications, ideal for intensive data processing.

The integration between DAPHNE and HDFS is designed to support seamless handling of large datasets and distributed computing tasks. In terms of the workflow specification, the integration facilitates data upload in both the DAPHNE binary format and the widely used CSV file format. This flexibility ensures compatibility with different data sources. Moreover, it provides distributed read and write capabilities, allowing for efficient management of large datasets across multiple nodes.

DAPHNE is a system employed for distributed data processing and machine learning tasks, optimized for high performance on big data workloads through its distributed computing architecture. After data processing is complete, the data save functionality enables users to

store the processed results in either DAPHNE binary format or CSV format, depending on their requirements. It also supports distributed write operations, offering users the ability to specify output destinations, whether that be HDFS or local storage.

The implemented methods underpin this integration. The system supports reading and writing in DAPHNE's native binary format with optimized efficiency. Similarly, it handles CSV file formats for input and output operations, maintaining compatibility with standard data formats. The integration also includes distributed read/write operations for both DAPHNE binary and CSV formats, ensuring scalable performance for large datasets in distributed environments. This combination of features makes the DAPHNE-HDFS integration highly effective for big data processing tasks.

We deployed HDFS on a private cluster of one coordinator and eight worker nodes, each equipped with 8 CPUs. The system was tested with both read and write operations to HDFS using a CSV matrix file of 430MB in size. The results, shown in Figure 1 showed that read performance consistently improves when utilizing HDFS, taking full advantage of its distributed architecture for efficient data access. Additionally, write performance improves as more nodes

are added to the system, demonstrating the scalability of HDFS when handling large datasets in distributed environments. More HDFS benchmarking results can be found in D4.4.

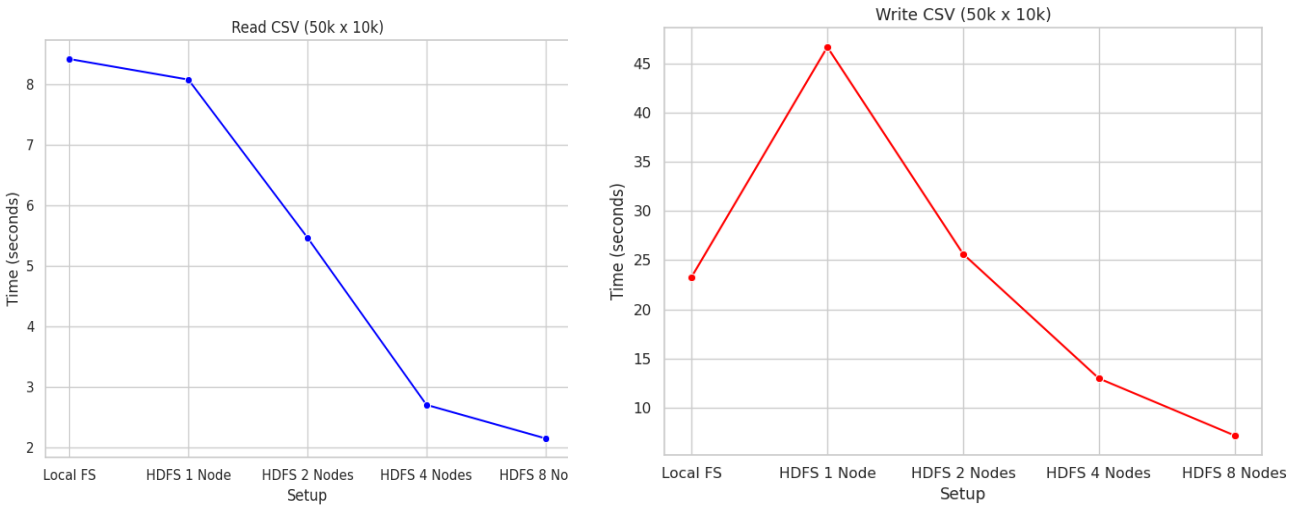


Figure 2: Read and Write Performance utilizing HDFS

Lustre Integration

Lustre[1] is a high-performance, open-source parallel distributed file system designed primarily for large-scale computing environments. It is widely used for applications requiring extensive data processing, such as scientific research, AI, and big data analytics. Lustre is renowned for its scalability, with the ability to handle petabytes of data and thousands of clients, making it suitable for high-performance computing (HPC) clusters and supercomputing environments.

The Lustre filesystem is built on a client-server architecture, split into specialized components that manage metadata and storage for high efficiency and scalability. Key components are:

- **Metadata Server (MDS)**: Manages metadata operations (e.g., directory structure, permissions).
- **Object Storage Server (OSS)**: Handles the actual file data, storing it across one or more Object Storage Targets (OSTs).
- **Management Server (MGS)**: Central repository for configuration data, which is shared among all Lustre clients.

[1] https://www.lustre.org/

- **Clients**: Endpoints that mount the Lustre filesystem, appearing as a POSIX-compliant mount point on the client OS.

These are pictorially described in Figure :

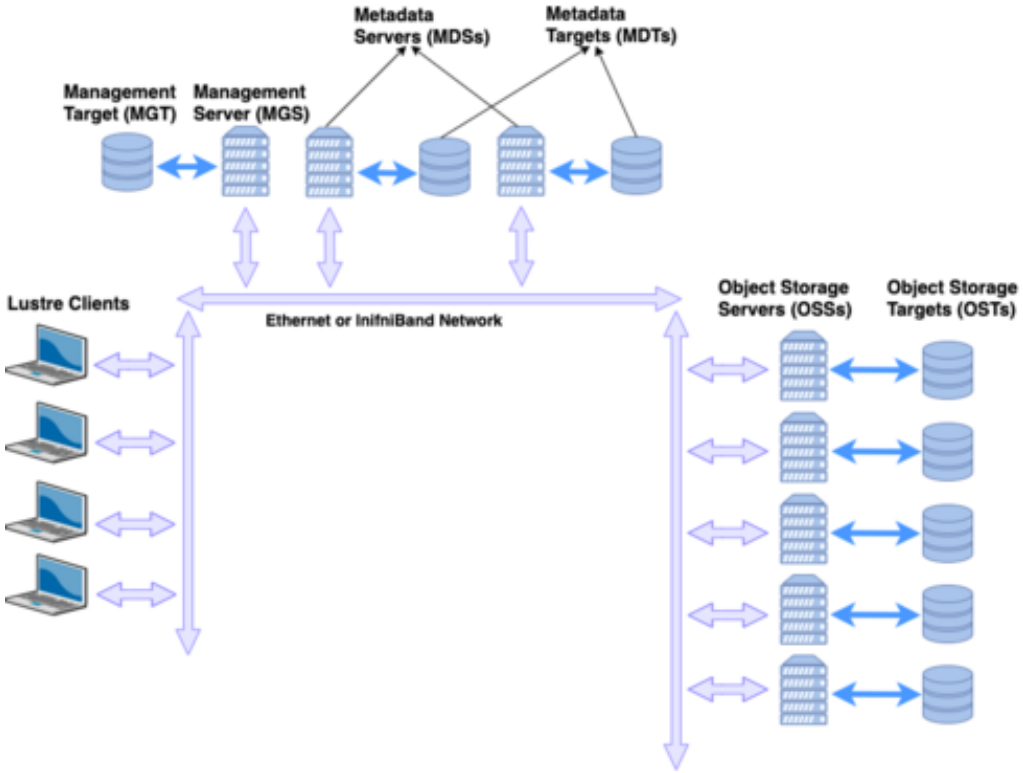


Figure 3: Lustre FS System Architecture

Lustre achieves performance through *data striping*, where files are divided across multiple Object Storage Targets (OSTs) in chunks called stripes based on parameters like `stripe_count` and `stripe_size`. The stripe count determines how many OSTs will be used to store the file data, while the stripe size determines how much data will be written to an OST before moving to the next OST in the layout. As an example, consider the file layouts shown in Figure [2] for a simple file system with 3 OSTs residing on 3 different OSS nodes.

[2] https://wiki.lustre.org/Understanding_Lustre_Internals

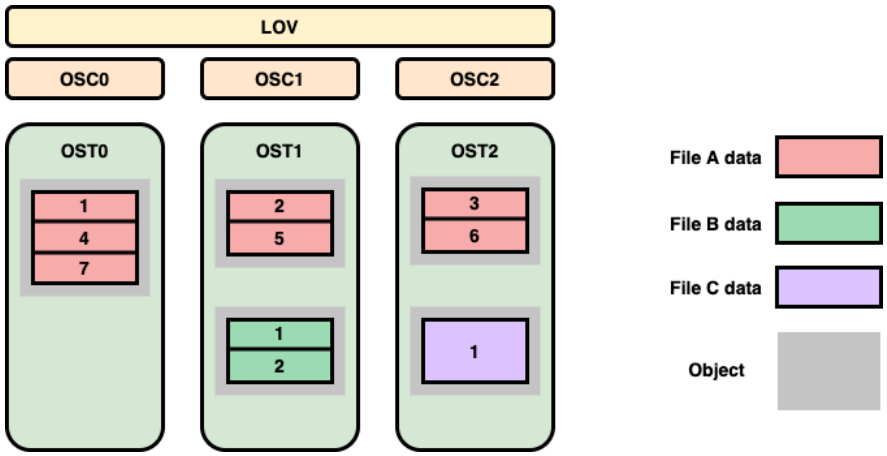


Figure 4: Normal file striping in Lustre

For integration with Daphne, a Lustre client is mounted and a Daphne container deployed on top, allowing Daphne kernels to interact with Lustre without embedding Lustre client dependencies directly into the container. The `liblustreapi` C API facilitates communication, allowing Read and Write operations on CSV files and serialized Daphne objects for both local and distributed runtime environments. However, CSV files require padding due to distributed operations, while serialized objects do not.

The prototype also allows parallel processing by assigning file segments to different workers based on calculated offsets, supporting synchronous gRPC as the backend for distributed operations. Ongoing experiments will attempt to measure performance across different Lustre configurations, comparing Lustre and HDFS performance. Additionally, the effect of aligning Lustre's stripe configuration with worker segments will be explored in order to investigate if positioning Daphne containers on OSTs improves performance.

3.4.1 NUMA-Aware Data Placement

NUMA-aware data placement is an optimization technique used in systems with Non-Uniform Memory Access (NUMA) architectures. In NUMA systems, memory is divided among different nodes (typically CPUs or CPU cores), and each node has its own local memory. Accessing local memory is faster for a CPU than accessing memory located on a different node, which incurs higher latency and lower bandwidth. NUMA-aware data placement thus organizes data in memory so that it is located as close as possible to the CPU or thread that will use it. This minimizes remote memory accesses, improving data access speed and overall system

performance. In terms of how memory is allocated (i.e., Memory Allocation Policies) we can briefly define the following NUMA policies:

- *Preferred Node:* Memory is allocated from a preferred node, but if that node's memory is exhausted, other nodes are used.
- *Interleaved Allocation:* Memory is spread evenly across nodes, balancing memory load and bandwidth but potentially increasing access latency.
- *Local Allocation:* Memory is allocated on the same node as the CPU that requested it, which helps minimize remote memory access.

We have taken advantage of the fact that many operating systems, like Linux, provide options to control memory allocation policies to make applications NUMA-aware. Our study examines DAPHNE's matrix processing capabilities in NUMA systems, focusing on performance across matrix types and configurations on two systems: an Intel(R) Xeon(R) Gold and a QEMU virtualized environment, both over an AMD EPYC server. Through various NUMA policies and memory placements, experiments were conducted using Linux NUMA tools, such as `numactl` and `numastat`, to observe performance behaviors in dense and sparse matrices, within multi-node NUMA environments. This setup allows insights into how memory allocation and CPU locality interact in NUMA-aware systems.

On the Intel Gold server (memory allocation fitted within a single NUMA node), tests with sparse matrices, particularly for the PageRank algorithm, showed that performance remained largely consistent across NUMA configurations when data was confined to a single NUMA node. The default Linux policy provided optimal or near-optimal results, with only marginal improvements under the `` `--interleave=all` `` and `` `-m0` `` policies. This consistency suggests that PageRank on sparse matrices is relatively unaffected by NUMA policies as long as the data fits within a single NUMA node.

For dense matrices on the Intel Gold server, however, performance significantly improved under specific NUMA configurations. The `` `-m0` `` policy, which aligns memory allocation with the local CPU, led to notable gains by reducing access latency and enhancing locality, making it ideal for dense data where frequent memory accesses benefit from minimized latency. Similar observations were made with the connected components algorithm. Here, the `` `-N0 -m0` ``

policy, which coordinates both memory allocation (`-m0`) and thread execution (`-N0`) within the same NUMA node, achieved a 15% performance improvement. This setup maximizes memory-thread locality, particularly important for dense matrix computations where latency-sensitive operations are common.

In the QEMU virtualized environment, which required memory spanning across two NUMA nodes, different policies yielded varied results. For sparse matrices running the PageRank algorithm, the `--interleave=all` policy provided improvements, achieving approximately 8% better performance by reducing NUMA misses. This configuration, which distributes memory across nodes, minimized contention issues that would otherwise degrade performance when data is forced across multiple nodes. Conversely, executing in a single node in QEMU led to performance drops due to memory contention, underscoring the importance of interleaving for sparse matrices in multi-node setups.

Dense matrices also benefited from `--interleave=all` in QEMU, as this policy helped distribute memory load evenly across nodes, reducing contention and improving runtime. The connected components algorithm showed that dense data in particular benefited from interleaving, with a 10% performance gain over the default policy. The findings indicate that in environments where memory must span multiple NUMA nodes, interleaving policies can mitigate NUMA overhead effectively, enhancing performance for dense matrices where cross-node memory access is more frequent.

Overall, results show that sparse matrices generally exhibit stability across NUMA configurations, especially when contained within a single NUMA node, and tend to benefit minimally from NUMA policies. Dense matrices, however, show greater sensitivity to NUMA-aware optimizations. In case of ample memory compared to the data size, aligning memory and thread locality with `-N0 -m0` optimized dense matrix performance, while in a more memory-constrained environment, interleaving policies were particularly effective in reducing latency for dense workloads.

These findings underscore the importance of NUMA-aware configurations tailored to the type of matrix and the specific memory topology of the environment. While interleaving improves

performance in dense, multi-node cases, default NUMA policies are often sufficient for sparse matrices, particularly for algorithms like connected components on QEMU, where additional interleaving may add unnecessary overhead.

3.4.2 Filetype Support

The Coordinate List (COO) sparse matrix representation was integrated into the DAPHNE runtime, enhancing its ability to handle sparse matrices efficiently. The COO format, unlike traditional dense or CSR (Compressed Sparse Row) formats, is especially suited for ultra-sparse matrices by storing only the non-zero values along with their row and column indices, thus optimizing memory usage. The implementation involved creating a `COOMatrix` class with attributes to manage matrix values, row, and column indices. Key methods include:

- Getters and Setters: For accessing and modifying matrix elements.
- Aggregation Kernels: New aggregation methods to handle operations across rows, columns, or the entire matrix.
- Element-wise Operations: Support for unary and binary element-wise operations on COO matrices.
- Random Matrix Generation and Transposition: For efficiently generating random COO matrices and performing transpositions.

The TDMS (Technical Data Management Streaming) file format was also integrated into the DAPHNE project, enhancing its data processing capabilities. TDMS is widely used in engineering and manufacturing for high-performance data acquisition, often handling large-scale datasets. This supports DAPHNE's goal of robust data processing, enabling it to handle more complex, large-scale datasets efficiently, particularly in technical and engineering domains.

TDMS files are organized in a three-level hierarchy (file, group, and channel) to manage large datasets effectively. Each file contains groups, which contain channels where the actual data is stored. Channels store the raw data, while metadata provides structure. TDMS files are composed of segments with three parts—Lead In, Metadata, and Raw Data. This layout allows for efficient data storage and access. Segments store metadata on the file structure and indexing, enabling quick retrieval of specific data segments.

For implementing TDMS reader methods, we utilized a C++ library to read TDMS files into DAPHNE's structures (DenseMatrix and Frame). This involved parsing TDMS data and mapping it to DAPHNE matrices or frames, organizing data by group and channel.

For TDMS writer methods: The writers convert DAPHNE matrices and frames into TDMS files by assigning a single group with multiple channels (one for each matrix column). Each data column is assigned a unique channel to facilitate structured data storage.

3.4.3 DAPHNE Runtime Prototype

Finally, we provide access to the Final DAPHNE Runtime prototype, publicly available in the DAPHNE development repository. Guidelines for building and running DAPHNE are included in this period's deliverable (D4.4), along with examples of using different storage backends when executing Daphne scripts.

3.5 WP5 Scheduling and Resource Sharing (UNIBAS) [M1-M48]

A key component of vectorized execution in local and distributed runtime environments is hierarchical scheduling across multiple nodes, heterogeneous hardware devices, and threads per device. The related design and decisions are closely related to WPs 3 and 4, with a specific focus on deployment environments and workloads (IDA pipelines, task-parallel loops and operator pipelines), task partitioning, queue management, and scheduling algorithms, as well as data and task placement for data-parallel processing of fused operator pipelines and kernels. Initial work laid the foundation of common terminology, devised, and materialized the scheduler design in deliverable D5.1 [10], as well as developed and integrated a library of alternative scheduling primitives (e.g., static and self-scheduling) for exploratory experiments into the vectorized execution engine, described in deliverables D5.2 — Initial Scheduling Prototype and D5.3 — Improved Scheduling Prototype.

Sparsity exploitation is a major trend across the software/hardware stack from ML algorithms, over ML systems, to the underlying hardware. Sparse data and operations in turn have challenging runtime characteristics due to irregular structures and skew. For that reason, hierarchical scheduling and task planning is a strategic objective because advanced scheduling

algorithms can yield significant performance improvements by mitigating the resulting load imbalance across workers and worker hierarchies. Additional work outside the prototype also investigated additional distribution primitives, collective operations (e.g., MPI, gRPC), parameter servers, and similar distribution strategies. All these topics are brought together under holistic hierarchical scheduling in deliverable D5.4.

3.6 WP6 Computational Storage (ITU) [M1-M48]

The integration of computational storage in DAPHNE requires the availability of devices supporting code offload, from the hosts where DAPHNE is run. We thus consider a form of data staging, where data is moved from archival services or object storage to the HPC cluster or cloud instance where integrated data pipelines are run. We consider that this data staging is orthogonal to DAPHNE's integrated data pipelines. We review existing tools for managing this form of data placement and discuss how encrypted data is handled.

WP6 started with a review of the state of the art and of the gaps that exist in the area of computational storage (deliverable D6.1 [11]). The NVME standard, the standard protocol for modern storage, was extended with a computational storage command set in January 2024. This standard relies on a protocol for downloading and executing functions on computational storage.

The Delilah prototype we described in D6.2 [12] proposed a similar protocol. We have thus been able to analyze the benefits and pitfalls of the standard based on the lessons we learned with our Delilah prototype. More generally, our end-to-end experiments with the Delilah prototype enabled us to discuss the nature of downloadable computational storage functions in the context of integrated data pipelines, also in comparison with other accelerators like GPUs and FPGAs explored in WP7.

3.7 WP7 HW Accelerator Integration (TUD) [M1-M48]

The integration of hardware accelerators into DAPHNE, as originally discussed in D7.1 [13], was further advanced and the design refined as well as initial operations were accelerated using GPU and FPGA as described in D7.2. [14] In the domain of GPU processors, the supported operations range from common unary and binary arithmetic over linear algebra to machine learning specific kernels commonly applied in neural networks. Additionally, supporting a certain type of hardware accelerator brings along the need for kernels that deal with context creation and device initialization, to keep track of the handles to their API and other specifics.

The memory management aspects of these devices are wrapped inside their respective classes and data handles and details of its representation is organized in so-called meta data object. This structure enables fine grained data placement decisions. For GPU computing, we rely on the CUDA API and the hardware that supports it to a large extent. However, an initial integration of OneAPI from Intel has been done to further extend the list of supported devices in this category and not solely rely on the products of a single vendor. This initial integration also drives the documentation efforts of extensibility to guide potential third parties in the endeavor of adding functionality to DAPHNE.

The second major family of hardware accelerators, namely FPGA, has also seen the successful development of important operations like quantization or general matrix multiply (GEMM) and an initial integration of an Intel Stratix based accelerator. Further efforts of more specialized work dealing with SIMD exploitation through a SIMD abstraction library, performance models and code generation has been conducted and documented in D7.2 [14] and D7.3. In particular, the further development of the SIMD abstraction library (virtual vector library), which not only maps to various SIMD extensions of general-purpose CPUs but also to Intel FPGA cards using OneAPI, should be emphasized. More details are reported in D7.3, which mainly summarizes our achieved results for code generation.

In order to simultaneously exploit these heterogeneous devices, we started to add the functionality of running fused pipelines through our vectorized execution engine on them. This integration not only enables us to further exploit available resources but also gives us the opportunity to add tuning knobs for scheduling and load balancing where extra care needs to be taken to cater to every device's needs. Regarding input and output, as the utilization of

accelerators usually implies a certain cost of pushing the task to the compute units and pulling the result back to main memory.

Finally, the fourth report D7.4 summarizes our work and achieved results with regard to multi-device operations and data placement on multiple homogeneous and heterogeneous devices. In particular, we discuss three different scenarios, each focusing on a different kind of multi-device setting. The first scenario focuses on accelerating relational data processing with the combined usage of thread-level and data-level parallelism (SIMD) on general-purpose CPU. Here, we introduce a novel, unified memory access pattern and evaluate the combined usage on different data layouts. In the second scenario, we investigate the co-processing capabilities of CPU and GPU using OneAPI for the relational data processing, while the third scenario investigates matrix operations in a multi-GPU environment. The achieved results for the multi-device setting are promising and developed concepts should be further pursued.

3.8 WP8 Use Case Studies (KAI) [M1-M48]

DLR Use Case:

DLR worked at developing a framework for large scale processing of satellite data with applications in planet scale LCZ classification. The developed tools allow the user to specify arbitrarily large geographical areas and process them patch-wise. Arbitrary pipelines for processing are supported. The pipelines are executed on a patch level. For example, the LCZ pipeline works by retrieving all Sentinel-2 data that overlaps with a given patch over the user specified time period, constructing a cloud free composite image based on the algorithm that is described in the next paragraph and then running a deep learning inference model on the composite image which returns LCZ classification for the given patch. We then reconstitute the LCZ classified patches into the LCZ classification of the whole user specified area. The framework is fault tolerant in the sense that if at any stage a pipeline or the software handling the execution of pipelines fails, it can be resumed from the failed step (after fixing the source of failure), the already completed work being preserved.

A critical part of the LCZ classification pipeline is the cloud removal algorithm. Cloud removal is important since cloud cover makes it impossible to determine that LCZ class of the area

underneath the cloud. It works by identifying cloud free pixels in a time series of images and then computing a statistic called the medoid (a multi-dimensional generalization of the median) over all cloud free pixels in the time series. Pixels here are 12 dimensional vectors instead of the more familiar R,G,B pixels of computer graphics. Hence the medoid is computed in 12 dimensions. The advantage of this method is that medoid represents a real pixel that is actually present in one of the images in the time series. Whereas if we were to use a statistic such as the mean, the values of the 12 bands present in the pixel would be averages that don't exist in the actual recorded data. This can lead to issues since the deep learning model that does LCZ classification is trained on real (non-averaged) data. We have used DAPHNE in an attempt to accelerate the medoid calculations which are slow due to the fact that they involve measuring the distances in twelve dimensional spaces for each pixel in the image to the corresponding pixel in every other image in the time series. Hence this operation scales quadratically with regards to the number of images involved. And we want as many images as possible to increase chances of cloud free data. We have converted the NumPy implementation to an implementation using DAPHNE matrix operations. We have also measured and compared the performance and presented the results in deliverable 8.2. Eventually, in D8.3 we recap on all use cases, give detailed information about the benchmarking setup and report on the benchmarking results of the various use cases for the DAPHNE system infrastructure.

IFAT Use Case:

The semiconductor manufacturing case study primarily focuses on enhancing productivity and minimizing downtime in ion implantation processes. The researchers developed a novel scheduling approach that integrates equipment condition-derived constraints to mitigate setup costs, resulting in a significant reduction of equipment downtime by over 100 hours annually.

In terms of data engineering and preprocessing, the study adapted methods to create an anonymized dataset, which was published on the Zenodo research repository. This dataset aids in developing predictive models aimed at reducing downtime associated with unsuccessful setups. The modeling and evaluation phase involved expert-guided feature selection and physics-based feature engineering, alongside the analysis of various metrics like confusion matrix, F1 score, precision, and recall. A supervised learning pipeline was established using built-in sensors and process target data to predict setup costs. These predictions are incorporated into scheduling systems as additional constraints to enhance equipment stability and utilization

through proactive dispatching adaptations. The models are regularized to minimize frequent shifts in predictions, thereby reducing re-scheduling efforts.

The study also integrated Machine Learning Operations (MLOps) principles, emphasizing containerization and orchestration for stable operation, which involved splitting preprocessing, modeling, and postprocessing into separate microservices.

Performance benchmarking compared the DAPHNE stack's runtime efficiencies and accuracy enhancements against a Python implementation. Although DaphneDSL uses less CPU and memory than Python, it has a longer runtime. Ongoing optimizations are expected to improve DaphneDSL's performance significantly.

Looking ahead, the researchers plan to finish deploying the productive solution in the real-world manufacturing environment and verify the anticipated improvements in uptime and runtime in DaphneDSL's decision tree implementation.

KAI Use Case:

In the fourth and final project year of DAPHNE, we were able to completely migrate our use-case pipeline to the DAPHNE system implementation. The required technical work is documented in D8.3: we have been able to read our measurements (180 csv-files holding about 17 GiB of data) into the DAPHNE system, perform necessary preprocessing steps and applied our line simplification algorithm. The result data is written to file afterwards.

We then compare the DAPHNE implementation to the base implementation in Python (which is using numpy-arrays) in terms of runtime and memory utilization:

- Runtime of DAPHNE is about 3.16 times faster than the Python implementation
- Memory usage of DAPHNE is stable (no memory leaks) and below the memory usage of Python

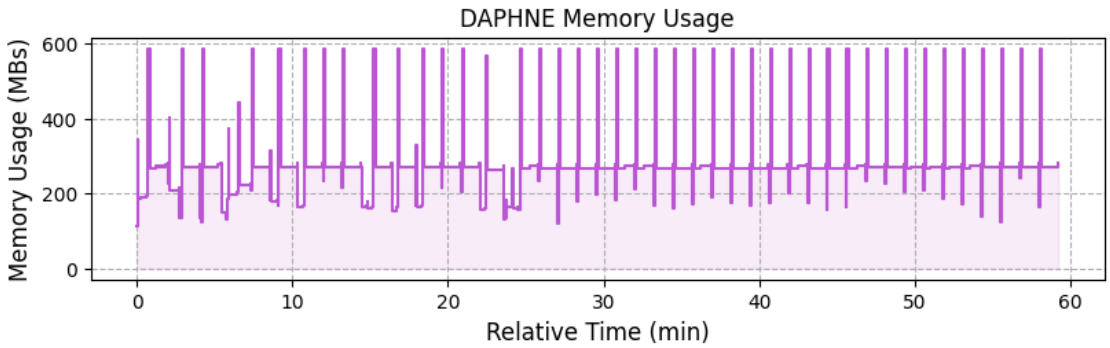


Figure 5: DAPHNE Memory Usage

Furthermore, in the 4th year of the project, we have also continued to work on the 2nd KAI pipeline: "Remaining Useful Lifetime Prediction". The initial work has been prepared within a Master thesis project by Mohamed Ghoneim. We have followed up the topic and a second Master thesis has been carried out by Alpertunga Ertin. Within this project, we modeled and trained a RUL prediction pipeline using PyTorch and inferred the learned model parameters to the above-mentioned dataset using DAPHNE. In this use-case however, the performance of the DAPHNE system takes about 1.45 times the runtime of the Python base implementation.

AVL Use Cases:

Within the Automotive Vehicle Development Case Study "Ejector geometry optimization for fuel cells", the focus was directed towards the design of experiments (DoE) workflow. Daphne Library (DaphneLib) has been integrated into the Python Machine Learning workflow at the prediction function stage. A benchmarking configuration has been set up and benchmarking results have been collected. The evaluation addresses the performance comparison between the stand-alone python implementation and the DaphneLib supported workflow. Details can be found in D8.3.

In the second case study "Virtual Prototype Development", a demonstrator (which can create artificial training data and running a Gaussian process regression) has been created, analyzed and its bottlenecks have been identified. The pipeline has been substantially reworked, improved and fully automated so that no manual input is further necessary. The revised pipeline creates realistic synthetic training data autonomously, without manual intervention during pipeline execution. A benchmarking setup that focuses on three elements of the pipeline (crossover, post-processing and visualization) has been set up. Relying on this setup, the

DaphneLib utilization and its effect of pipeline performance have been analyzed. Results can be found in D8.3.

WP8 created common terminology and a joint understanding of requirements via regular joint meetings for in-depth discussions of the individual use cases (including knowledge sharing between partners), the use case descriptions, and ML pipeline implementations. A major outcome of these discussions is the use case pipelines documented in D8.1 [15], which serve as example top-down use cases for the DAPHNE system infrastructure and real-world benchmarks. During these discussions, we already identified future work for improvements of the individual pipelines and relevant measurements to quantify the use case improvements achieved through DAPHNE (in terms of development productivity and runtime performance for training and scoring).

3.9 WP9 Benchmarking and Analysis (HPI) [M1-M48]

Throughout the DAPHNE project (M1-M48) we surveyed existing DM, HPC, and ML System benchmarks (M1-M12), defined the DAPHNEBench Benchmark (M9-M30), and developed an internal benchmarking and profiling toolkit (M19-M48). We discussed our progress on these tasks and coordinated our collaboration with the other project partners remotely in regular work package meetings and all-hands meetings, as well as, in person in general assembly and dedicated use case meetings.

To report our progress to the EU we summarized our work in the review meetings and submitted deliverables 9.1 to 9.4. In the following, we give an overview of our progress throughout the four years of working on the DAPHNE project by summarizing our work on the four deliverables.

D9.1 - A Survey of Benchmarks from DM, HPC, and ML Systems [M12]

In our survey on Big Data, HPC, and ML benchmarking frameworks [16], we have identified a need for a benchmarking framework that will encompass and measure the performance of Integrated Data Analysis (IDA) pipelines as defined in Figure . The findings in deliverable 9.1 were based on the lack of convergence in DM, HPC, and ML benchmarking frameworks. We

proposed a framework that combines metrics and measurement aspects from the three domains.

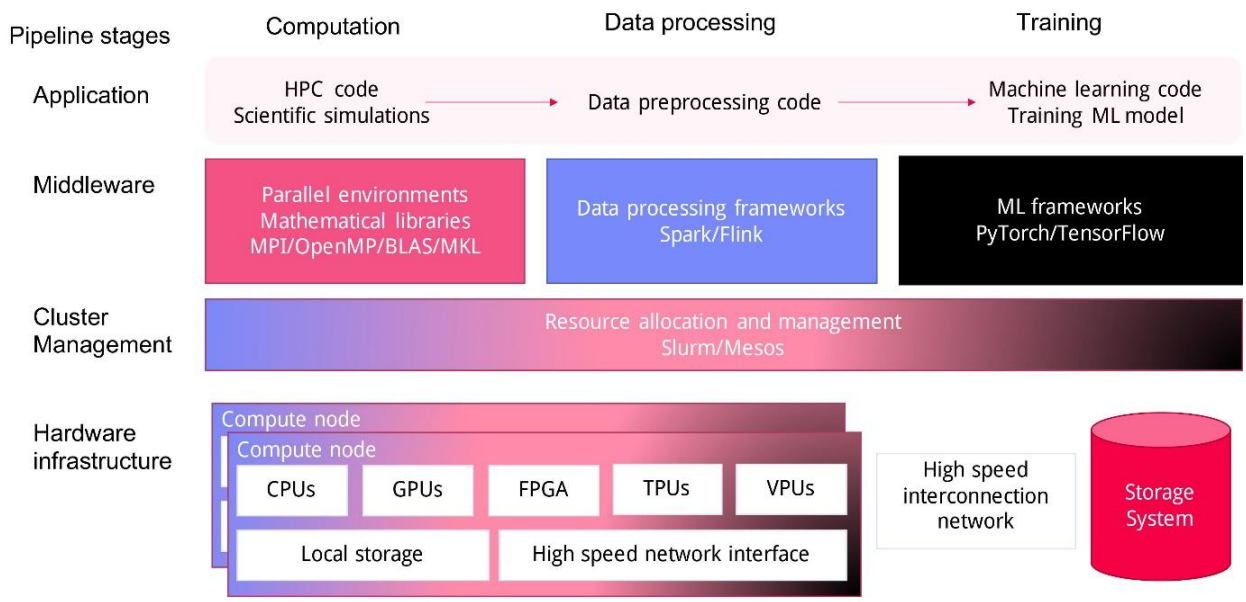


Figure 6: Integrated Data Analysis (IDA) Pipeline

In this deliverable, we discussed the state-of-the-art of BD, HPC, and ML benchmarks. We summarized a representative selection of some of the most used benchmarks and classified them under the light of a feature space composed of purpose, stage, metric, and convergence, as well as from the perspective of a proposed Integrated Data Analysis architecture. In Table 1, Table 2, and **Fehler! Verweisquelle konnte nicht gefunden werden.**, we show the three groups of benchmarking frameworks, the evaluated system aspects, as well as the quality metrics.

Aspect	Benchmarks	Metrics
Data collection	BigBench, Graphalytics	throughput, latency
Data analysis	HiBench, BigBench, BigDataBench, Graphalytics	resource consumption (CPU, memory, I/O), execution time
Data storage	BigBench, YCSB	throughput, latency

Table 1: Big Data Benchmarks

Aspect	Benchmarks	Metrics
Compute	NPB, HPCG, SPEC	speedup, throughput, node communication latency, data access latency
Data	NPB, CORAL-2	
Network	NPB, SPEC	
Memory	CORAL-2, UEABS, HPCC	
I/O	NPB, CORAL-2	

Table 2: HPC Benchmarks

Aspect	Benchmarks	Metrics
Data Preparation & Cleaning	CleanML	execution time, resource consumption (CPU, GPU, memory), difference in model performance
Model Training	CleanML, MLPerf, DeepBench, LEAF, Fathom	execution time, resource consumption (CPU, GPU, memory), model performance, time-to-accuracy
Model Inference	MLPerf, DeepBench	execution time, resource consumption (CPU, GPU, memory), inference performance

Table 3: ML Benchmarks

We observed that the evaluated benchmarking frameworks cover a wide range of purposes and stages of BD, HPC, and ML systems. However, even if modern hybrid-systems become

more common, benchmarking systems are still not fully capable of targeting those data analytics systems and can impact only small parts of the IDA pipelines.

D9.2 - Initial Benchmark Concept and Definition [M24]

In Deliverable 9.2, we outlined a set of requirements necessary to define an initial concept of the benchmarking framework (see Figure). The focus of the proposed framework is to capture the complete pipeline lifecycle, covering multiple benchmarking aspects and abstraction levels of IDA pipelines. To evaluate the complete pipeline lifecycle, we defined metrics for tracking the end-to-end performance, as well as metrics that cover different runtime aspects. The list of supervised and valued metrics is shown in Table *4*.

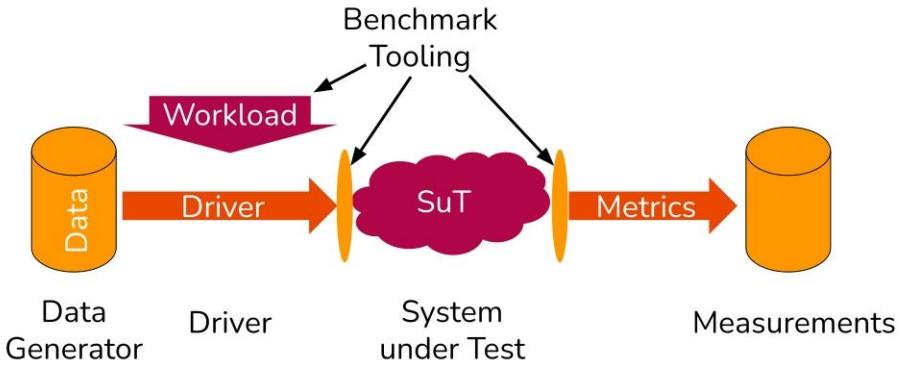


Figure 7: An Outline of a Benchmarking Framework

We defined the system under test to provide different levels of abstraction. In the benchmarking framework definition, we cover scenarios in which the SuT can be a single method call, a single pipeline stage, as well as a complete pipeline. We capture the diversity in IDA pipelines by implementing diverse workloads based on DAPHNE use cases, as well as open-source workloads. Specifically, we used the workloads developed in DAPHNE Use Case 1 - Earth Observation, and Anomaly Analysis Use Case provided by hardware manufacturer Backblaze [4].

Supervised metrics	Time, Memory, Energy, CPU consumption, Latency, Throughput
Valued metrics	Confusion matrix, Hyperparameter influence, Epochs to accuracy, Epochs to loss

Table 4: Benchmarking Metrics

We defined a data model that supports evaluating the heterogeneity of IDA pipelines. The data model allows us to capture performance and runtime characteristics in the several pipeline dimensions. We have summarized the definition of the benchmarking toolkit in deliverable 9.2 [17].

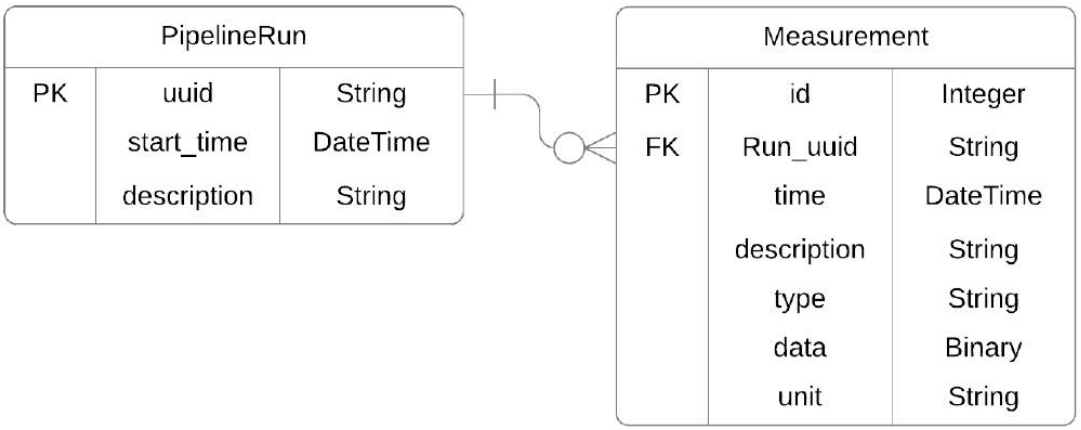


Figure 8: Data Model for Monitoring of Experiments

D9.3 - Initial Benchmarking Prototype [M30]

In deliverable 9.3, we implemented an internal benchmarking toolkit that is able to cover a wide variety of metrics. To easily integrate the measurement of any metric without significant change to the original pipeline used for benchmarking we follow a decorator-based design.

```
(base) ilint@ilint-Precision-5550:~/HPI/DAPHNE/daphne-pipelines$ umlaut-cli daphe_pipelines.db
? Please select one or more uuids.  done (3 selections)
? Please select measurement types corresponding to uuids.  (<up>, <down> to move, <space> to select, <a> to toggle, <i> to invert)
  Available types for uuid a62127b9-422c-4bb2-b7a8-e665df508ffa
  ●cpu
  ○energy
  ○memory
  ○power
  ○time
  Available types for uuid d8534210-2c89-45a2-a415-f86bfcc9fddd
  ●cpu
  ○energy
  ○memory
  ○power
  ○time
  Available types for uuid feb6b495-53ff-4c78-a7a1-65518fa949ed
❯ ●cpu
  ○energy
  ○memory
  ○power
  ○time
```

Figure 9: Visualization of the Command Line Interface

To measure all metrics of interest we integrated widely used libraries and tooling such as the Python *time* library [5] to use Python's performance counter for time-based metrics, *psutil* (python system and process utilities) [6] for memory and CPU metrics, and the *pyRAPL* library to measure energy and power [7]. All measurements are logged into a database backend and are interactively accessible via an easy-to-use command line interface that automatically provides plots for easy analysis of measurements shown in Figure 9: Visualization of the Command Line Interface

To automate benchmarking results for some real workloads as easily as possible, we started implementing first versions of scripts that automatically run real-life use cases provided by our project partners. For now, we focused on the Earth Observation use case and the Back-blaze Anomaly Analysis use cases.

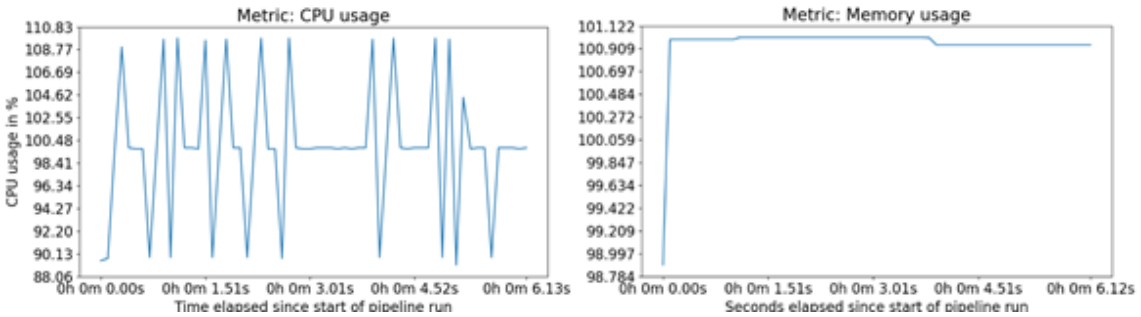


Figure 10: Visualizing CPU and Memory Usage for a Single Pipeline

In this deliverable, we demonstrated and discussed our benchmarking toolkit with work package partners at the DAPHNE use case meeting in Graz where we also showed some initial results of bench-marking the DAPHNE workloads written in DAPHNE's domain-specific language (DSL) as well as in DAPHNE's proprietary version of its Python interface DaphneLib. Some of the initial measurements are shown in Figure 10: Visualizing CPU and Memory Usage for a Single Pipeline

D9.4 - Final Prototype of the Benchmarking Framework [M48]

In D9.4 we present the finalized version of our internal benchmarking and profiling toolkit UMLAUT. Through communication with the industrial use case partners and the rest of the DAPHNE consortium, we defined a set of improvements for the initial prototype. These included additional features, metrics, as well as improvements to the overall user experience regarding installation, usability, and visualization.

To quantify the resource overhead of incorporating UMLAUT in a new environment, we implemented the three simple benchmarking use cases of sleeping, sorting, and matrix multiplication with UMLAUT and added them to our repository. Since all use cases have predictable benchmarking results, they allow users to validate UMLAUT's functionality and give insights on setup-specific benchmarking overheads.

To limit the installation overhead when using UMLAUT, we provide three docker containers. One UMLAUT-only container with all UMLAUT dependencies installed to run and benchmark Python scripts, a second container that has UMLAUT and DAPHNE installed to benchmark DAPHNE DSL and DaphneLib scripts, and a third container that has UMLAUT, DPAHNE, and GPU drivers installed to benchmark GPU accelerated workloads. Using these containers allows us to easily run and benchmark all Python, DaphneDSL, and DaphneLib pipelines provided by the use case partners.

We extended the set of supervised metrics by the four GPU-specific metrics listed in Table 5 and implemented them using the NVIDIA Management Library (NVML). Using these metrics allows an UMLAUT user to track the GPU utilization, GPU memory, GPU time, and GPU power usage.

Metric name	Description
GPUMetric()	Measures the GPU utilization in percent using nvmlDeviceGetUtilizationRates()
GPUMemoryMetric()	Measures the GPU memory usage in MB using nvmlDeviceGetMemoryInfo()
GPUTimeMetric()	Measures the execution time using CUDA Events. There is usually a small difference between this and time.perf_counter(), which matters for quick methods.
GPUPowerMetric()	Measures the power usage of the GPU in Watts.

Table 5: Extended Benchmarking Metrics

To improve the monitoring of DAPHNE DSL scripts, we have implemented subprocess tracking, allowing us to have detailed measurements for the DSL workload when executed from a Python environment. This enables users to have a finer granular overview of the DSL experiments compared to the prototype implementation of the framework.

We demonstrate how to use UMLAUT for benchmarking plain Python pipelines, DAPHNE DSL implementation of the IFAT Semiconductors use case, and the GPU-accelerated KAI Material Degradation use case. The resources for both use cases were provided by the use case partners.

For a better user experience, we changed UMLAUT's interface for plots from MatplotLib to Plotly and implemented the functionality to provide custom names in the decorator functions. As shown in Figure 11: Interactive Plotting in UMLAUT with Plotly, this allows users to identify relevant measurements easily and to hide irrelevant data for a better overview.

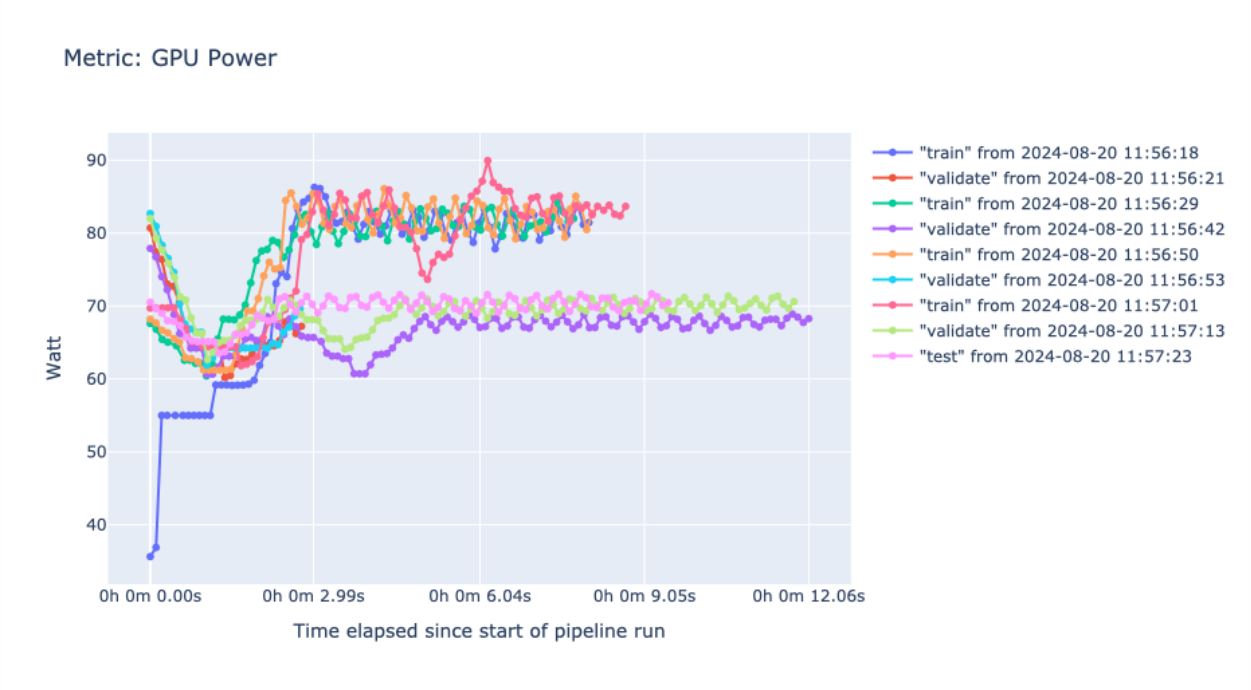


Figure 11: Interactive Plotting in UMLAUT with Plotly

3.10. Dissemination and Exploitation (KNOW) [M10-M48]

The primary objective of dissemination and exploitation is a broad and open sharing of the project results via publications and talks, dedicated networking efforts, and an open-source reference implementation. In project year 3 these activities have been re-defined in D10.1 Refined Dissemination and Exploitation plan [18] and have been carried out accordingly.

In the first task of this WP T10.1 - which is the only active task until M48 - Continuous Dissemination via Publications and Talks [M10-M48, Lead: KNOW, Participants: all, Effort: 13PM] - complementary to the work in the technical WPs that perform dissemination via scientific publications, which is interleaved with the actual research, 81 publications and 102 talks have been organized or co-organized are being listed and further communicated via the EU portal and conveyed to the public via our DAPHNE website [2].

The dissemination tracking document is updated every 3 months asking project partners to provide details concerning their DAPHNE-related activities and update the listed items. The publications and talks are inter-linked according to the open-source principle. The organization

of the dissemination tracker is time-consuming, but necessary as all relevant dissemination and communication activities must be recorded.

Thus, the current overview of our publications and talks throughout the project runtime till now (M1-M36) shows 81 publications and 102 talks. They directly link to the source material. We have achieved to promote DAPHNE in the central venues of data management, such as VLDB, HiPEAC and Euro-Par as well as workshops organized by the European Big Data Value Association (BDVA). KNOW are an active member of the interplay between EU research policy and research organizations through our leading role in BDVA. KNOW former CEO has been on the board of directors (BoD) of BDVA since 2016, granting KNOW access to the policy making level. Her mandate was taken over by our current CSO Roman Kern. KNOW also actively participate in BDVA taskforces.

In addition to the listed publications and talks we have also connected to similar projects such as MARVEL, EVEREST, or eFlows4HPC synergizing within ML/DL systems and system support. In detail, in a reciprocal process we have been exploring architectures for gathering heterogenous data, language abstractions, intermediate representation, methods for extreme-scale analytics, e.g. combination of ML models, simulations and subsequent data analysis in different Use Cases, standardized interconnection methods, e.g., runtime integration, HPC libraries as well as data fusion and data integration technologies.

As complementary dissemination and exploitation measures to maximize impact - in addition to publications and talks and exploring synergies by reaching out to similar projects - we have contributed to benchmarks and standards and have implemented a release timeline to the open-sourced DAPHNE reference implementation with the latest 0.3 release in August 2024. We have started to facilitate real-world data or datasets with similar characteristics to simplify reproducing our experimental results.

Eventually, as planned we have regularly updated our DAPHNE website https://DAPHNE-eu.eu [2] which communicates the project idea and objectives. The website gives a general overview of the project, showing latest news, listing all consortium partners, displaying a team photo of our personal gathering for the TEC-Use Case Workshop in Graz in September 2023, presenting portrait photos of the consortium partners, referring to the EU's Horizon 2020 research and innovation program, displaying a regularly updated list of publications and talks including the public deliverables, presenting the DAPHNE use cases, and showing a contact form for visitors

to get in touch. The website also invites to visit the open-source community on GitHub and follow DAPHNE project on social media. DAPHNE is active on LinkedIn. We consider these social media engagements via LinkedIn not as core but still as must-haves to promote our project and easily link to our target users and project partners. Within the last project year, we have increased the quantity and quality of our social media posts as well as our efforts towards project exploitation. Therefore, we have created a product portfolio element for DAPHNE to make it more accessible to a broader audience. In this portfolio we describe the advantages of DAPHNE to potential customers. Regarding visuals, we have a DAPHNE poster that can be adapted easily.

Regarding DAPHNE major target groups, our dissemination efforts address system or data engineers that are designing and operating robust infrastructures, and application users that are mostly concerned with productivity and accurate predictions. The open-source strategy has been mobilized to attract target users and invite them to give feedback.

4 References

[1] D1.1 Project and risk management plan [confidential]

[2] DAPHNE website https://DAPHNE-eu.eu

[3] D2.1 Initial System Architecture [public] https://DAPHNE-eu.eu/dissemination/

[4] D2.2 Refined System Architecture [public] https://daphne-eu.eu/wp-content/uploads/2022/08/D2.2-Refined-System-Architecture.pdf

[5] D3.1 Language Design Specification [public] https://DAPHNE-eu.eu/dissemination/

[6] D3.2 Compiler Prototype [public] https://DAPHNE-eu.eu/dissemination/

[7] D3.3 Extended Compiler Prototype [public] https://daphne-eu.eu/dissemination/

[8] D3.5 Final Compiler Prototype [public] (to be submitted in accordance with this report/M48)

[9] D3.4 Compiler Design and Overview [public] https://daphne-eu.eu/dissemination/

[10] D5.1 Scheduler design for pipelines and tasks [public] https://DAPHNE-eu.eu/dissemination/

[11] D6.1 Report on search space analysis, automatic capability configuration [public] https://DAPHNE-eu.eu/dissemination/

[12] D6.2 Prototype and overview of managed storage tiers and near-data processing [public] https://DAPHNE-eu.eu/dissemination/

[13] D7.1 Design of integration HW accelerators [public] https://DAPHNE-eu.eu/dissemination/

[14] D7.2 Prototype and Overview HW Accelerator Support and Performance Models [public] https://daphne-eu.eu/dissemination/

[15] D8.1 Initial pipeline definition all use cases [public] https://DAPHNE-eu.eu/dissemination/

[16] Ihde, Nina, et al. "A Survey of Big Data, High Performance Computing, and Machine Learn-ing Benchmarks." Technology Conference on Performance Evaluation and Benchmarking. Springer, Cham, 2021.

[17] D9.2 Initial Benchmark Concept and Definition [public] https://daphne-eu.eu/dissemination/

[18] D10.1 Refined dissemination and exploitation plan [confidential]