H2020-JTI-EuroHPC-2019-1

Project no. 956748

ADAPTIVE MULTI-TIER INTELLIGENT DATA MANAGER FOR EXASCALE

D7.1
Applications requirement definition.

Version 1.0

Date: September 29, 2021

Type: Deliverable
WP number: WP7

Editor: Adalberto Perez
Institution: KTH

Project co-funded by the European Union Horizon 2020 JTI-EuroHPC research and innovation programme and Spain, Germany, France, Italy, Poland, and Sweden		
Dissemination Level		
PU	Public	√
PP	Restricted to other programme participants (including the Commission Services)	
RE	Restricted to a group specified by the consortium (including the Commission Services)	
CO	Confidential, only for members of the consortium (including the Commission Services)	

Change Log

Rev.	Date	Who	Site	What
1	21/04/21	Jesus Carretero	UC3M	Document creation.
2	20/07/21	Adalberto Perez	KTH	Fist draft outlining the document structure.
3	17/08/21	Adalberto Perez	KTH	Second draft moving application description and analysis from the WP7 partner contributions.
4	20/08/21	Shahbaz Memon	FZJ	Formalized the document's final structure and streamlined a couple of sections.
5	24/08/21	Adalberto Perez	KTH	Implemented comments produced from initial review. Document ready for the WP7 internal review.
6	27/08/21	M. Torquati, A.R. Martinelli, B. Cantalupo, M. Aldinucci, V. Bono	CINI	Definition of the SHAnalytics application.
7	28/08/21	M. Torquati	CINI	Introduced some syntactical amendments.
8	30/08/21	Marek Justyna	PSNC	Updated description of the Life Sciences application.
9	31/08/21	R. Sedona, S. Sharma	FZJ	Updated description and application's I/O of the Remote Sensing application.
10	01/09/21	S. Kirti, S. Pandey	FZJ	Proofread the Document.
11	3/09/21	R. Montella, D. Di Luccio	CINI	Updated description of the Environment application.
12	14/09/21	Ramón Nou	BSC	Review of the deliverable.
13	17/09/21	Ahmad Tarraf	TUDA	Review of the deliverable
14	20/09/21	Adalberto Perez	KTH	Corrections from review #1 and #2 for the general document and the turbulence application.
15	22/09/21	Andrea Piserchia	CINECA	Corrections from review #1 (added scalability concerning Car–Parrinello module) and #2 for the Quantum Espresso application.
16	27/09/21	Ariel Oleksiak	PSNC	Final check and minor updates with regard to the Life Sciences use case
17	27/09/21	R. Montella, D. Di Luccio, L. Marcellino, G. Giunta	CINI	Final check and minor updates with regard to the Environment use case
18	28/09/21	Adalberto Perez	KTH	Final check of the complete document.

Executive Summary

The ADMIRE project aims to establish control in HPC systems, balancing storage and computations to maximize overall system performance. One of the key aspects of the ADMIRE project is co-design, where a variety of partners and their applications will collaborate to develop and test the ADMIRE technologies to improve I/O systems in preparation for Exa-scale systems.

On this deliverable, we documented the scientific use of each of the applications involved in the co-design process. Furthermore, details of their implementations and computational workloads were provided. However, the main objective of the deliverable was to provide the application requirements as inputs to the technical work packages for the development of the ADMIRE technologies. For this purpose, we collected the requirements employing an application I/O analysis that includes I/O behavior, APIs, and its characterization. The information is initially exposed on a "per application" basis, organized on its usability as collected from a requirement template develop in the work package. It is then summarized and analyzed to determine possible priorities.

A relevant outcome of the activities described in this document was the set of requirements and the methodology developed to collect them. The data collected will be used throughout the ADMIRE project.

The document is divided into the following chapters:

1. Introduction.
2. Application Description.
3. Application Characterization and Requirements.
4. Requirements Summary, Insights and Future Work.

Contents

Chapter 1

Introduction

The ADMIRE project aims to establish control in HPC systems, balancing storage and computations to maximize overall system performance. The co-design process within the ADMIRE project intends to analyze a set of scientific applications and provide design inputs into the technical work packages apart from preparing the applications to run with the future ADMIRE technologies. The general structure of the work plan is given in Figure 1.1, where a close communication between the applications and each of the work packages can be observed.

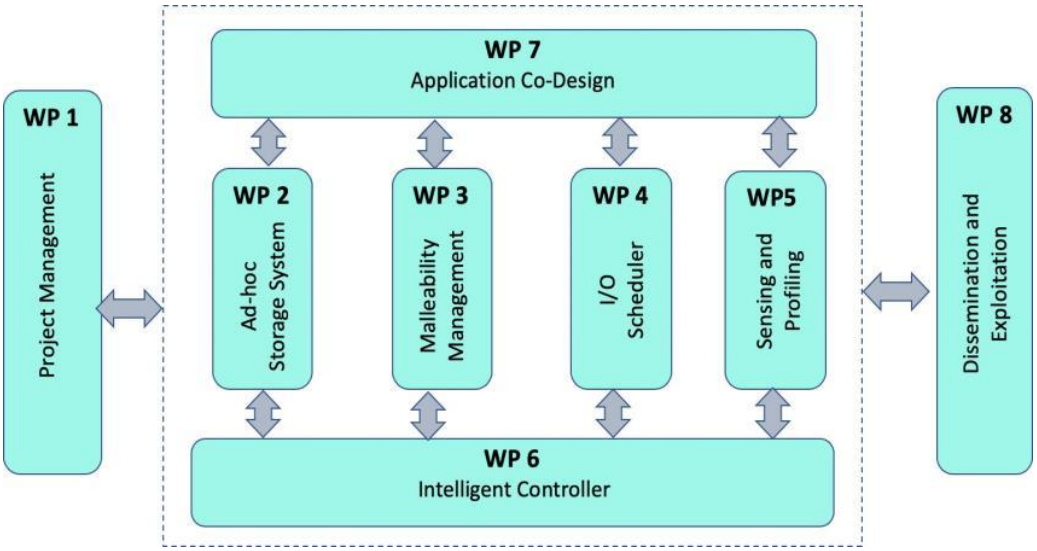


Figure 1.1: ADMIRE Work-plan.

The primary objective of this document is to provide the initial input from use-cases to the project. Specific objectives are briefly described in the next section.

1.1 Objectives

- Document the usability and scientific context of the applications defined in the ADMIRE portfolio.
- Define both a methodology and guidelines for applications characterization to gather significant information for the project.
- Document the applications' requirements based on their characterization.
- Analyze the requirements and identify common necessities among applications to establish priorities.

1.2 Approach

For this document, we define application requirements as a set of conditions that the ADMIRE technologies are restricted to throughout the project. To obtain the aforementioned set of conditions, all the applications in the portfolio were analyzed and their current I/O performance was documented. The collected information represents requirements to the technologies designed and implemented during the ADMIRE project. This deliverable intends to reflect the results of Task 7.1 (see Section 1.3) by employing the following actions:

- Defining a methodology for capturing and recording relevant information in a template defined in work package 7.
- The characterization of each application is grouped according to the relevance of the information and serves as a requirement for the different ADMIRE subsystems.

1.3 Tasks Associated With the Deliverable

Task Number	Task Description	Planned Activities	Performed Activities
T7.1	Application case studies requirement collection.	Definition of relevant use cases and applications. Documentation of requirements for the ADMIRE system architecture.	Definition of relevant use cases and applications. Documentation of requirements for the ADMIRE system architecture.

Table 1.1: Tasks associated with the deliverable.

1.4 Document outline

The document is divided into three main chapters:

- Chapter 2: Describes the scientific context of each application providing details on how they work.
- Chapter 3: Organizes the information collected from each application and provides some initial analysis.
- Chapter 4: Summarizes and analyzes the information collected in Chapter 3 by providing insights and defining the set of requirements from the applications.

Chapter 2

Application Description

The information contained in this section aims to provide context regarding the scientific field in which each of the applications involved in the co-design process of the ADMIRE project is used, apart from giving more details on how the applications perform their work. This is done by providing information regarding how the applications are built, what sort of computational task is performed during their execution, and how an application interacts with others codes or applications in case their respective pipelines. The current software in the co-design process are:

- **Environment (A1 - ENV):** Marine and weather forecasts and simulations.
- **Molecule Simulation - Quantum-Espresso (A2 - QE):** Describes complex electronic interactions.
- **Turbulence Simulation - Nek5000 (A3 - NEK):** Simulations of large-scale turbulent flows.
- **Remote sensing (A4 - RS):** Analysis of remotely sensed images.
- **Life Sciences - SRRF (A5 - SRRF):** Live-cell Super-Resolution Microscopy.
- **Software Heritage Analytics (A6 - SHA):** Software stack that enables Data Analytics on Software Heritage.

2.1 Environment (A1 - ENV)

The Environment Application produces operational weather and marine forecasts and/or on demand ad-hoc environmental simulations for scenarios and what-if analysis. The application is currently deployed on a dedicated HPC cluster for operational purposes, named “BlackJeans”. The results of the environmental forecast are published once a day and freely available at meteo@uniparthenope as interactive thematic maps, timeseries, tables, and open data format as well. Open data is accessible using standard protocols widely used in the computational environmental science community (e.g. OPEnDAP and WMS) [10, 13, 16].

The application workflow starts when the NCEP Global Forecast System (GFS) initial and boundary conditions are available for download. Once the GFS data is downloaded, the computation workflow engine DagOnStar [14, 18–20] begins performing the data pre-processing needed by the Weather Research and Forecasting (WRF) numerical model. This model is a globally used, next-generation, mesoscale numerical weather prediction system designed for both atmospheric research and operational forecasting applications. It features two dynamical cores, a data assimilation system, and a software architecture supporting parallel computation and system extensibility.The model serves a wide range of meteorological applications across scales from tens of meters to thousands of kilometers.

The current model implementation is designed to reach a ground resolution of 1 km on the South of Italy ($d03_{WRF}$), 5 km on Italy and surrounding seas ($d02_{WRF}$), and 25 km on the Euro-Mediterranean area ($d01_{WRF}$). BlackJeans performs one run per day with data initialized at 00 Zulu, forecasting the next 168 hours. Every 24 simulated/forecasted hours, the results are published and made available to the users or to other simulation

models (air quality model, oceanographic model, etc.). This configuration is called "streaming forecast". Although each operational application run produces 168 hours of weather and marine forecasts, it recalculates the previous day (24 simulated hours) using GFS re-analysis as initial and boundary conditions restarting the simulation previously run. The aim of this approach is twofold: 1) using the re-analysis the simulation is driven by assimilated weather data; 2) restarting the previous re-analysis initialized simulation ensure coherent results for further processing.

The WRF raw results are moved to a high-performance accessible storage that is closely connected to the web servers. A more processed output, which is projected to regular latitude/longitude spatial domain, and enriched by diagnostic variables, is also moved. In the framework of the Environment Application, the WRF daily results are used for other models to produce marine and air quality predictions.

WaveWatch III (WW3) is a third-generation wave model developed at NOAA/NCEP in the spirit of the WAM model. It is an improvement of the models WaveWatch (developed at the Delft University of Technology) and WaveWatch II (developed at NASA, Goddard Space Flight Center). WW3, however, differs from its predecessors in many important aspects such as the governing equations, the model structure, the numerical methods, and the physical parameterizations. Starting from the latest versions, WaveWatch III is evolving from a wave model into a framework, which allows for easy development of additional physical and numerical approaches to wave modeling (https://polar.ncep.noaa.gov/waves/wavewatch/). As already done with WRF, three telescopic computing grids were configured: the ground resolution in the Mediterranean area ($d01_{WW3}$) was set to 9 km, the resolution for the seas surrounding the Italian peninsula ($d02_{WW3}$) was set as 3 km while the central and southern Tyrrhenian sea, east sector ($d03_{WW3}$), is covered by a resolution of 1 km. The data produced by the WW3 daily forecast are saved on a fast accessible storage and offered to the users with diverse and different services.

As is the case for WW3, the Regional Ocean Model System (ROMS) is offline coupled with WRF for wind friction and fed with initial and boundary conditions produced by the Copernicus European Project. ROMS is a free-surface, terrain-following, primitive equations ocean model widely used by the scientific community for a diverse range of applications. It includes several vertical mixing schemes, multiple levels of nesting, and composed grids (https://www.myroms.org). In the Environment Application, the configuration of ROMS includes the sole $d03_{ROMS}$ domain covering, at the resolution of about 160 meters, the geographic area between the southern Lazio Region and the northern Calabria Region spanning for more than 100 x 50 nautical miles. The three-dimensional predictions about the sea currents, the water temperature, the water salinity, and the sea surface height (about 2.5 TB of data per month) are stored for science and engineering usage and to force more coupled models.

WRF, WW3, and ROMS models generate what we call "first-level" prediction products. The workflow running on BlackJeans orchestrates the execution of models carrying out the "second-level" products: those models forecasting the weather, the sea waves, and the sea currents alongside other conditions are used to feed specific applications.

The ROMS model outputs are used to feed a pollutant transport and diffusion model named WaComM (Water Community Model) alongside the Campania Region coastal pollution emission sources database. WaComM [3, 4, 15] is a three-dimensional Lagrangian model, designed and implemented as an evolution of the Lagrangian Assessment for Marine Pollution 3D (LAMP3D) [2, 5]. It computes the transport and diffusion of pollutants for assessing the water quality near the mussel farms. In WaComM, several basic algorithms have been optimized and, to improve its performance in a High-Performance Computing environment, some features like restarting and parallelization techniques in shared memory environments have been added.

A brief description of the underlying mathematical model follows: pollutants are modeled as inert Lagrangian particles. No interactions with other particles or feedback are included in the model algorithm. The pollution sources are defined as a geographic location in terms of longitude/latitude, depth, the amount of Lagrangian particles released in one hour, and the emission profile that could be set statically or change during the simulation.

The WaComM system can be used in more ways: as a decision support tool, to aid in the selection of the best suitable areas for farming activity deployment, or in an ex-post fashion to achieve better management of offshore activities.

Within the ADMIRE project, we will contribute to the refinement of the new incarnation of WaComM

named WaComM++ (“plusplus”), which supports distributed memory (MPI), shared memory (OpenMP), and GPU (CUDA) parallelization.

The whole application is based on a workflow engine (DagOnStar in the production deployment) orchestrating diverse and different executables typed as follows:

- Initial and boundary conditions data downloaders: Executed when data is available, kick-off workflow branches.
- Data conversion and model couplers: Perform all the operations devoted to coupling the output of a model to the input to the next in the workflow model chain.
- Data renderers: Post-process the model output to produce a homogenized output file in NetCDF.
- “Legacy” models: Physic equation solvers, perform the actual performance demanding computation, “legacy” because developed by a wide community of field experts, the modification of the source code should be avoided.
- Models: physic equation solvers, perform the actual performance demanding computation, developed by the UNP research team: the source code can be modified within the ADMIRE project.

In order to estimate the workload, we consider the models and the resources involved in the production environment:

- WRF: Domain 01 (230 $\times$ 209 $\times$ 28 = grid points, computed for t time intervals, resolution: 25 Km), Domain 02 (361 $\times$ 336$\times$ 28 = grid points, computed for t time intervals, resolution: 5 Km), Domain 03 (301 $\times$ 306 $\times$ 28 = grid points, computed for t time intervals, resolution: 1 Km).
- WW3: Domain 01 (608 $\times$ 203 $\times$ 1 = grid points, computed for t time intervals, resolution: 0.09°), Domain 02 (486 $\times$ 353 $\times$ 1 = grid points, computed for t time intervals, resolution: 0.03°), Domain 03 (350 $\times$ 200 $\times$ 1 = grid points, computed for t time intervals, resolution: 0.01°).
- ROMS: Domain 03 (1134 $\times$ 1527 $\times$ 30 = grid points, computed for t time intervals, resolution: 160 m).
- WaComM++: about 10M particles computed t time intervals (it is a Lagrangian model, so we don’t have a spatial resolution).

Each model has a typical dominant kernel (for example WRF and ROMS the Navier and Stocks equation solver, the WW3 the random phase spectral action density balance equation for wavenumber-direction spectra solver), nevertheless, we will focus on WaComM++ which is characterized by the typical Lagrangian kernel (deterministic & stochastic particle movement). Analyzing the models’ production logs, the following rates can be estimated for each model:

- WRF: 75% computation, 25% I/O.
- WW3: 95% computation, 5% I/O.
- ROMS: 70% computation, 30% I/O.
- WaComM++: 60% computation, 40% I/O.

The WaComM++ rate is motivated by the hierarchical parallel approach used in which only the main process performs I/O operations.

2.2 Molecule Simulation (A2 - QE)

Quantum ESPRESSO (QE) is an integrated suite of open-source computer codes for ab initio quantum chemistry methods of electronic structure calculations and materials modeling at the nanoscale [7–9]. It is based on density functional theory, plane waves, and pseudopotentials. The code is widely used among the scientific community and in specific industries since it can address several computational material science tasks, e.g., ground-state calculations, structural optimization, ab initio molecular dynamics, potential energy surfaces, electrochemistry and special boundary conditions, response and spectroscopic properties, quantum transport, generation of pseudopotentials, etc. and can guide the rational new material design process.

To address part and/or all (depending on a generic user's requirement) different tasks, the application is constituted by a distribution of independent and interoperable codes/modules including a set of historical components and a set of plug-ins that perform more advanced tasks. It also includes several third-party packages designed to be interoperable with core components.

The main components of the QE distribution are designed to exploit the architecture of today's supercomputers characterized by multiple levels and layers of inter-processor communication. The parallelization is achieved using both MPI and OpenMP parallelization, allowing the main codes of the distribution to run in parallel on computing nodes with very good performance. The code is quite scalable and portable.

As an example the scalability of the QE Car–Parrinello module is shown in Figure 2.1 for a test case concerning a Car–Parrinello molecular dynamics simulation of the 5R7Y COVID protein. (Other scalability benchmarks regarding different QE modules can be found here). Scalability is quite good even if ideally one would like to have linear scaling. It is important to note that the effectiveness of parallelization is largely dependent upon:

1. the size and type of the system under study;
2. the judicious choice of the various levels of parallelization (QE offers several levels of hierarchically structured MPI communicators. The detailed description of the same can be found on the QE manual);
3. the availability of fast interprocess communications (or lack of it).

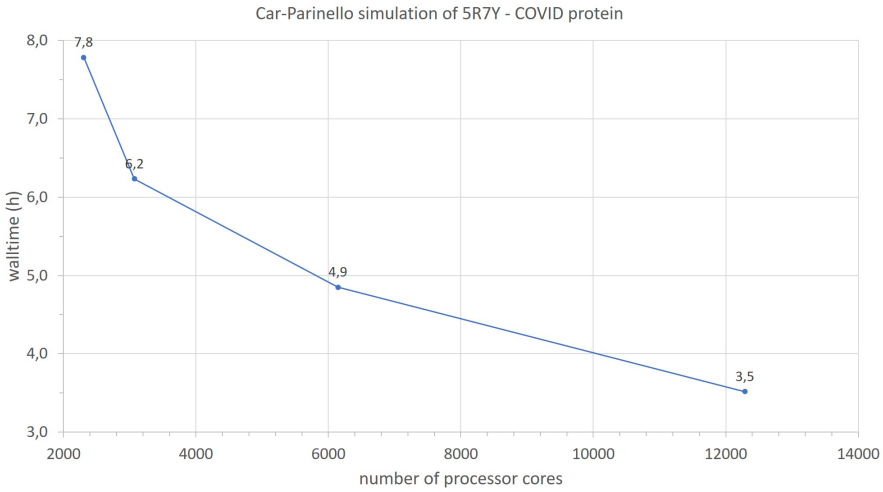


Figure 2.1: Scalability graph for the CP simulation of 5R7Y-COVID protein executed on the Marconi SKL cluster at CINECA.

Point 2 can be critical in terms of scalability since it can directly and/or indirectly afflicts "load balancing", required memory per processor, communication traffic, etc. Therefore, each different system under study should be specifically tuned by choosing the right pattern of MPI communicators performing accurate benchmark tests. Moreover, for each system there is an optimal range of number of processors on which to run the job; a too large number of processors will yield performance degradation. Finally, in order to increase scalability, it is often

convenient to further subdivide a pool of processors into "task groups". When the number of processors exceeds the number of FFT planes, data can be redistributed to "task groups" so that each group can process several wavefunctions at the same time. Further details are specified in the manual. In the context of the ADMIRE project the focus will be on the Car–Parrinello molecular dynamics simulation of zirconium oxide ZrO_2.

After the code is compiled (via Autotools, i.e. configure and Makefile and/or CMake), the binary executables are typically located in the "bin" directory. Then the generic user can run the desired executable (depending on the desired task) by providing a specific input file where the system intended to be simulated is specified (coordinates and other physical quantities) together with several simulation parameters. Specifically for Car-Parrinello molecular dynamics simulations (in this case the executable is named cp.x) input data for cp.x is organized into several namelists, followed by other fields ("cards") introduced by keywords (documentation of the required and optimal parameters provided in the input file are easily found via the internet and/or directly inside the QE folder). Output frequency can be directly tuned in this input file. Once the input file is ready, the program can be run via the command line.

Parallel execution is strongly system- and installation-dependent. Typically one has to specify:

1. A launcher program such as mpirun or mpiexec, with appropriate options (if any).
2. The number of processors, typically as an option to the launcher program.
3. The program to be executed (e.g., executable cp.x), with the proper path if needed.
4. Other QE-specific parallelization options, to be read and interpreted by the running code.

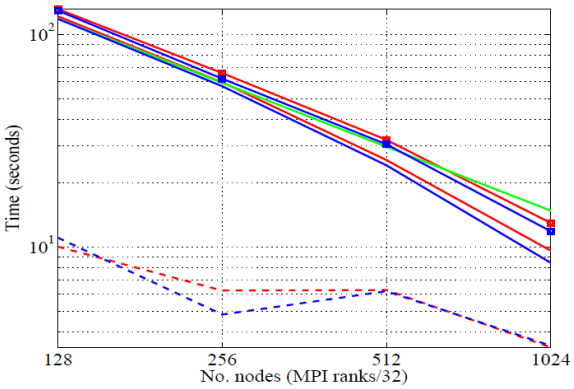
1 and 2 are machine-and-installation-dependent, and may be different for interactive and batch execution. 3 is also dependent on the users' specific configuration (shell, execution path, etc), whereas 4 is optional but essential for achieving good performance. Typically a job script file (e.g., for SLURM or PBS) is used to collect the modules to be loaded and specifies where the run command line.

The number of executables called is limited and they are typically stand-alone modules called by a "main executable" (e.g., cp.x) to perform specific tasks. There are no particular interaction within themselves that directly or indirectly afflict I/O. Therefore, the whole picture can be safely approximated as if there is only one executable that deals with all the operations associated with the execution, be it computations or storage operations.

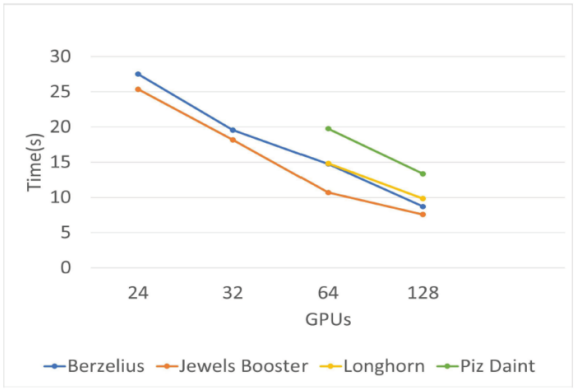
After the input file and pseudopotentials are given, the workload consists almost entirely of diagonalizing big matrices in order to solve the eigenvalue problem that arises from the Kohn–Sham formulation. This is a very computer-intensive task and is generally executed on parallel systems. Also, several discrete Fourier transformations are required since 3D FFT is needed in order to transform electronic wave functions from reciprocal to real space and vice versa. This task is accomplished by using fftw libraries that compute the discrete 3D FFT. The same is parallelized by distributing planes of the 3D grid in real space to processors.

2.3 Turbulence Simulation (A3 - NEK)

Nek5000 is an open-source CFD solver created at the Argonne National Laboratory [6] and based on the Spectral Element Method (SEM), i.e. a high order variant of the Finite Element Method (FEM). Because of these characteristics, a continuous Galerkin formulation is used, where the basis functions for the approximations are the Lagrange polynomials. The space discretization used is always based on the Gauss-Lobatto-Legendre (GLL) points, which combined with the Lagrange polynomials, gives good characteristics to the solver. Among the most important characteristics of the solver is the fact that, thanks to the formulation, equations can be solved locally in each element, hence, without the need for intermediate communication steps except when transferring data to other MPI ranks at specific times. On top of this, a matrix-free formulation is implemented, which means that operator matrices do not need to be constructed explicitly. This reduces the number of operations and memory needed for simulations. All these characteristics make the code very efficient and scalable. Figure 2.2 shows extracts of the scalability analysis performed to the CPU and GPU version of Nek5000 by [17] and [23], respectively. Note, however, that the GPU enabled code is still under testing and will not, at least initially, be considered for the ADMIRE project.



(a) CPU Strong scaling test of Nek5000 retrieved from [17]. The green line represents the linear scaling while the colored lines are the performance under different settings.



(b) GPU scaling test for the OpenACC enabled version of Nek5000 retrieved from [23]. Performance is tested on variey of systems as the line colors indicate.

Figure 2.2: Scalability of Nek5000.

The code is used for fluid dynamic simulations, which are essential for a large array of industries such as automotive, aerospace, energy, etc. The particular case to be profiled in this document is concerned with bent pipes, an essential component in machines and infrastructure. The purpose of this test case, is to identify the physical causes of oscillations produced in bent pipes as a consequence of the airflow through them. A schematic view of the case can be seen in Figure 2.3, taken from [12].

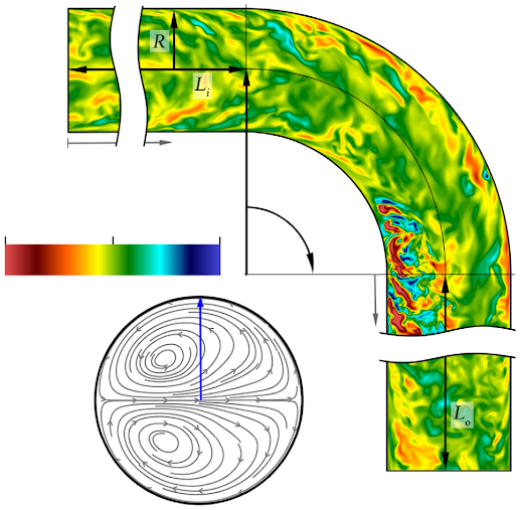


Figure 2.3: Turbulence case study.

The study of turbulent flows is very expensive due to the characteristics of the regime, which require the computational domain to be extremely fine to capture and fully resolve all the flow features. A case such as the turbulent pipe will require around 400.000 elements, each with 512 data points, i.e 204.800.000 grid points in the domain with multiple values in double precision are stored per gridpoint at multiple time steps in the simulation.

The application has a certain particularity, which is the need to compile when new settings need to be used (e.g. polynomial order, etc.). For this reason, the compilation process is included in the workflow. However, this is not I/O or computational intensive and can be generally done in log-in nodes on clusters. Three relevant folders need to be taken into consideration in the process. Initially, the compile folder and the Nek5000 source folder interact during the compilation where, for each new case, a «casename».usr must be included, which contains relevant information about the initial and boundary conditions of the physical case apart from other set-up instructions. Usually, a compile script is used to set up the environment variables needed for the executable

to be generated.

Once the executable is obtained, it must be placed in a folder that contains 2 particular files. A «casename».re2 file contains the information of the mesh. The second file is «casename».ma2 which contains indexing information for each node in the mesh and partitioning information for proper work balancing between MPI ranks. The application also counts with methods to perform dynamic partitioning at runtime. Figure 2.4 presents a schematic view of the process.

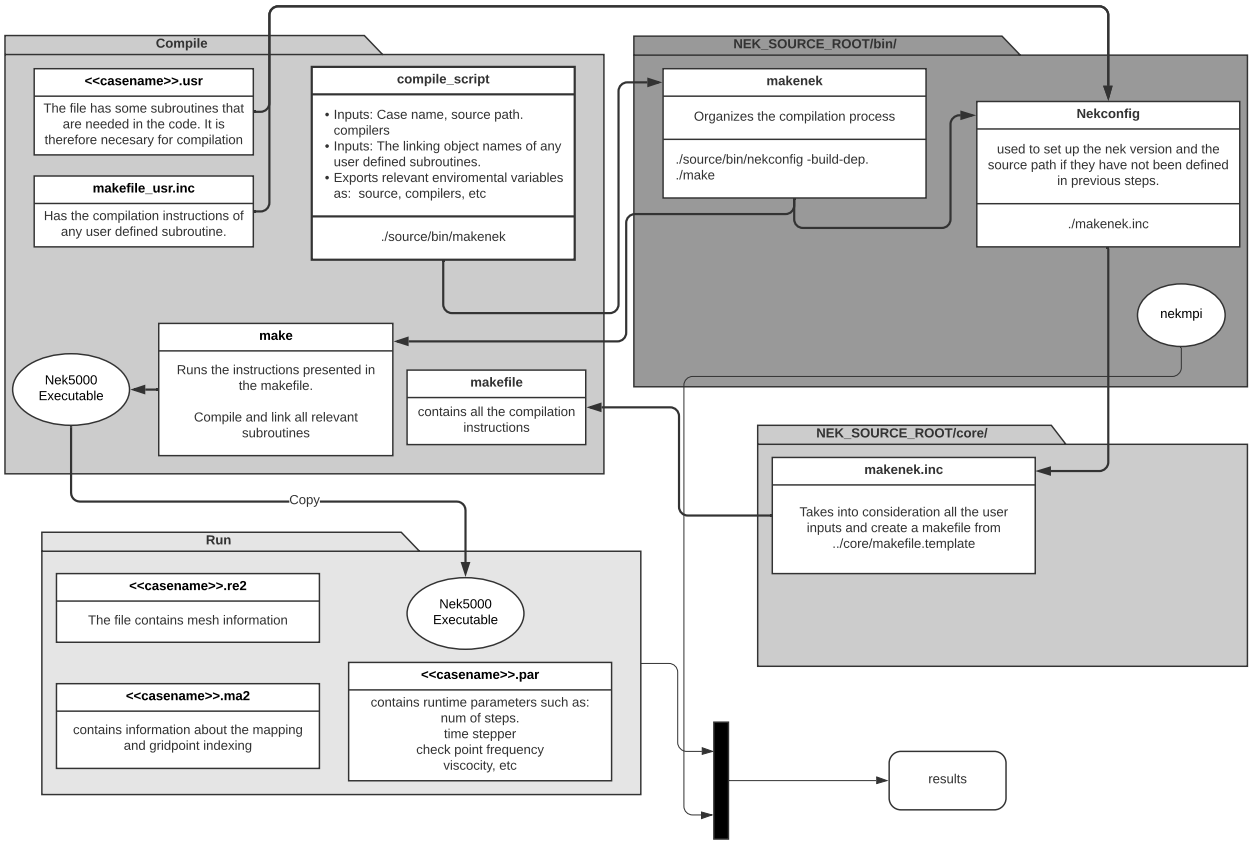


Figure 2.4: Nek5000 workflow.

To obtain the .re2 and .ma2 files, a mesh needs to be created, which is generally the most time-consuming pre-processing step in any sort of simulation. Nek5000 counts with a tool for meshing, however, it is mainly a solver and it accepts meshes that are created in specialized software such as gmsh among others subjected to a conversion step into the Nek5000 format employing internal methods.

Once the solver runs, the checkpoint and time step data are written into the file system. We call "time step data" the main outputs of the simulation, which are used in post processing and analysis stages. The checkpoints are recorded conditionally at user-defined intervals, and serve as a fault safety mechanism. Having obtained all the needed data, post-processing is done, where data is transformed and usually visualized through external tools such as VisIt or Paraview.

Nek5000 integrates the Navier Stokes equations over time using the Spectral Element Method (SEM), as mentioned before. The initial fields are read at the beginning of the execution and operations are made to calculate the relevant values in the new time step. The SEM discretization, like any finite element method, produces a set of matrix operators and variables such as the stiffness and mass matrices, which are used in computations with vector fields of one or more previous time steps (depending on the time discretization) to obtain the desired new values of the following time step. Due to the method, these operations take the

form of tensor products, which can in turn be expressed in terms of matrix-matrix multiplications. Due to this fact, the workload of the application is very compute-intensive and depends on the number of grid points present in the discretization, which would normally define the size of the matrices and vector fields, thus, defining the number of operations to be done. These activities are usually executed on parallel systems such as supercomputers, where each core receives a certain number of elements and makes the computations locally before communicating boundary points to the neighbors.

2.4 Remote sensing (A4 - RS)

The remote sensing application trains a multispectral (not only RGB channels but also infrared) ResNet Convulotional Neural Network (CNN) on BigEarthNet, a large remote sensing dataset, on top of performing a classification on a subset of the dataset. The classification problem is multi-label, meaning that more than one label can be associated with each sample.

The training data are loaded on each GPU in batches, iteration after iteration. At each iteration, Horovod, the framework for distributing the training of the DL model, takes care of synchronizing the gradients among the workers, and then an update of the weights is performed, so that the models on each worker are consistent.

The data are loaded by each GPU at the beginning of every iteration through the TensorFlow Dataset API. The computation of the gradients happens at first independently on each worker, and at the end of the iteration, the gradients are exchanged with a Ring-AllReduce algorithm, and the weights are updated consequently. The main kernel in the calculations performed is expected to be the the one containing the CUDA operations (matrix multiplications due to the large presence of convolutions) during gradient computation.

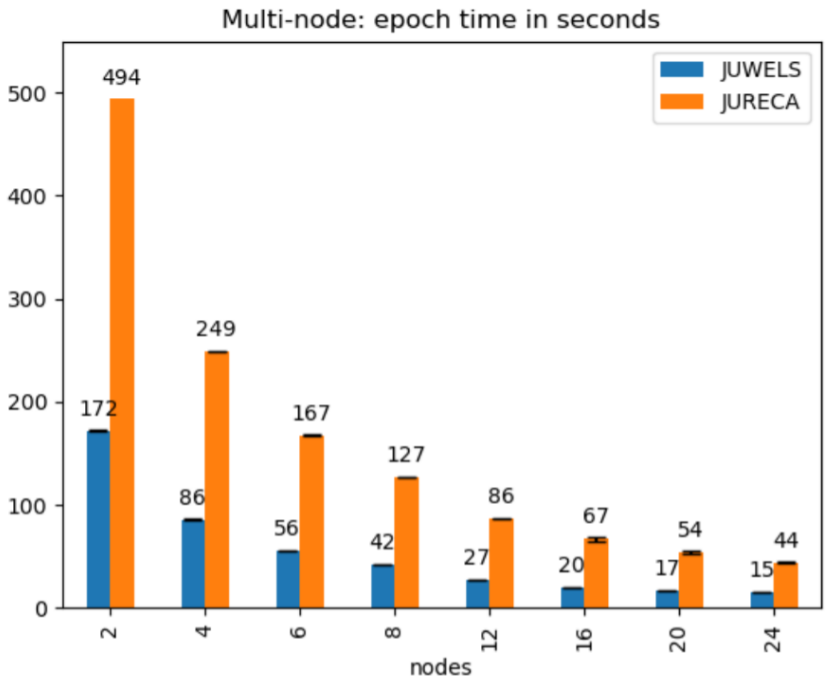


Figure 2.5: Training time per epoch [22].

The current application is a development of the work presented in [22]. The multispectral ResNet was deployed on multiple GPUs to reduce the time burden to train the model, keeping an efficiency above 90% up to 96 devices (Figure 2.5). A large portion of the study was dedicated to the analysis of the impact of the global batch size on the test accuracy. It has been noted that above a threshold of 8000 samples for the batch size, the generalization of the model tends to decrease. A number of strategies have been proposed, implemented, and adopted in order to reduce the impact of such issue. As our new code is implemented using the TensorFlow Dataset API instead of a custom dataloader, our interest is now devoted to understanding how scalable the code is in addition to profile the various phases of the training (data loading and computation) to check if further

tuning would be possible.

All the implementation is done in python, thus no stand-alone executable is compiled.

2.5 Life Sciences (A5 - SRRF)

Super Resolution Radial Fluctuations (SRRF pronounced as surf) is an algorithm for producing super-resolution images implemented in Java by Henriques Lab [11]. The input images come from fluorescent microscopy. Unfortunately, the fluorescent markers in tested samples can produce noisy images. In order to reduce the noise and increase the resolution of the final image, a sequence of images is captured (e.g. 100 frames). Then, all images are processed with SRRF to produce super-resolution output images. The application can be run as a plugin to ImageJ [21] application, which is a common tool used by many researchers. By default, SRRF is used together with a GUI, however a batch mode version has been also prepared by PSNC in order to allow easier usage with HPC systems (e.g. SLURM) when running computationally demanding tasks.

Figure 2.6 presents an example output of SRRF algorithm. The proceeded microscopy data comes from a mouse brain. The goal of such analysis is to reveal the population of cells involved in the pathogenesis of the polyQ neurodegenerative diseases. Moreover, further processing of these images is performed using machine learning algorithms to help the researchers in the analysis of such big sets of images.

The sequence of images is stored in a directory. The application reads all images at once and then starts processing (preferably on GPU with OpenCL). The output is a single image, which is saved to the local storage. The application runs with only one executable with the most I/O intensive part being the first phase of the program, which reads all images from the sequence.

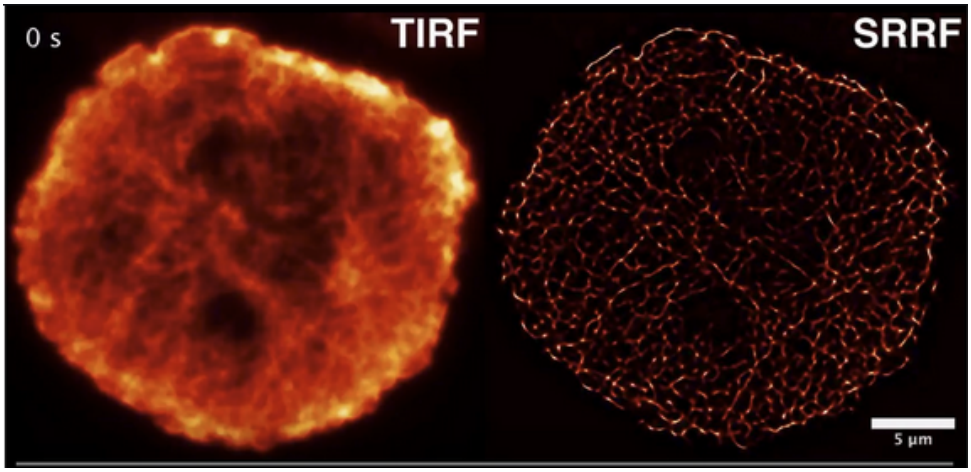


Figure 2.6: An example comparison of total internal reflection fluorescence (TIRF) and SRRF visualisation.

For the test case to be analyzed in ADMIRE, there will be approximately 5 TB of data to process, by executing multiple tests that might contain images with a total size ranging from hundreds of MBs up to a few GBs for a single run.

For benchmarks a representative example was chosen - one image. The representative image is 3.7 GB large. Figure 2.7 presents times measured when SRRF was executed only on CPUs. The performance and scalability are very poor. Over 2000 seconds are needed to proceed the whole image. The potential reason of poor scalability on CPUs might be probably the implementation of the program and nature of the language (Java). The primary goal of the SRRF developers was to run it on GPU, as GPU allows to perform these computations much faster, e.g. a single GPU requires only 180 seconds to proceed the same image, while on CPU computations can take over 2000 seconds. Further more, as presented on Figure 2.8, the scalability on GPUs is much better.

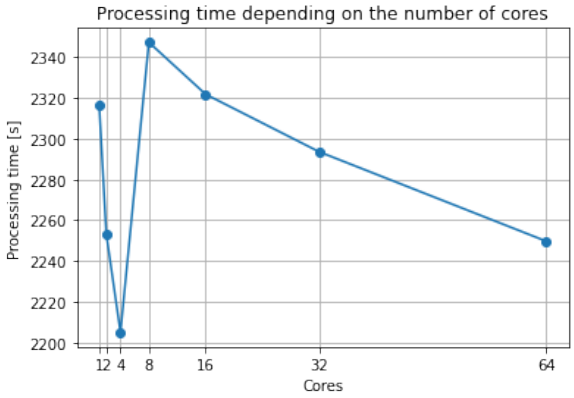


Figure 2.7: Benchmarks measured on SRRF using CPUs only

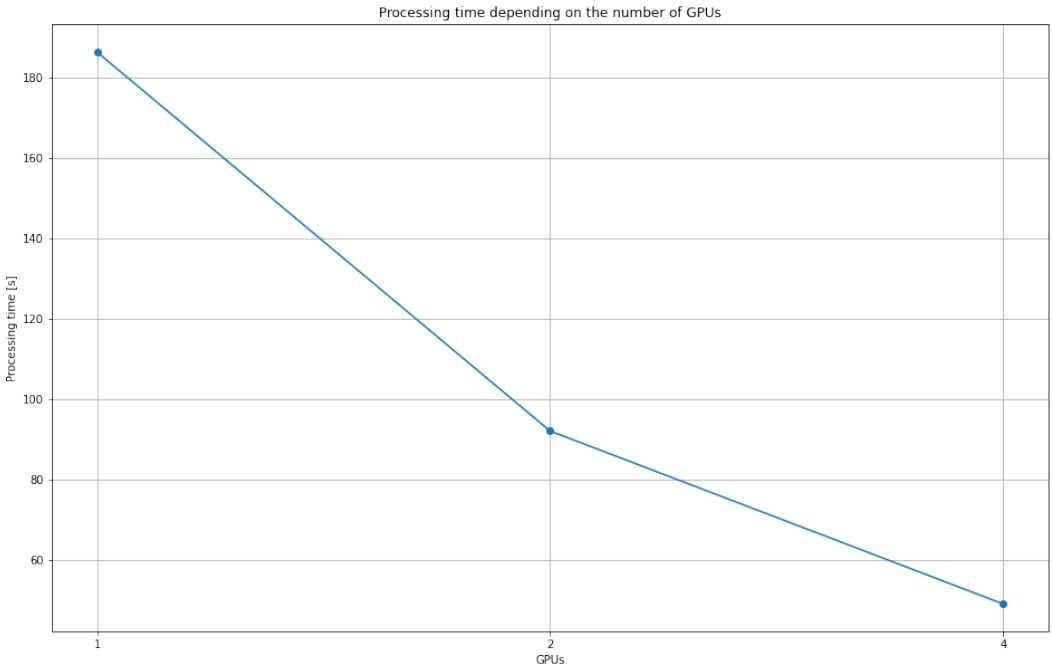


Figure 2.8: Benchmarks measured on SRRF using GPUs only

2.6 Software Heritage Analytics (A6 - SHA)

The Software Heritage (SH) project was designed to collect and preserve software in source code form, because software embodies our technical and scientific knowledge, and humanity cannot afford the risk of losing it. However, SH is designed to store data, not to analyze it robustly. In the interactive usage of SH (see Figure 2.9 top part), files are directly retrieved from the remote server via the Web API. The Software Heritage Analytics (SHAnalytics or SHA – A6), instead, exploits a local copy of the SH data and metadata, implements a batch extraction of the files to be examined and then produces a stream of them to be analyzed (see Figure 2.9 bottom part).

The SHAnalytics is a framework for running user-defined data analysis applications rather than a single application. It enables the execution of Stream Data Analytics applications on top of Software Heritage. User-defined analytics can be easily performed on the data files stored in the SH repository through the SHAnalytics framework. It presents a sequence of text files belonging to one or more software projects selected by the users

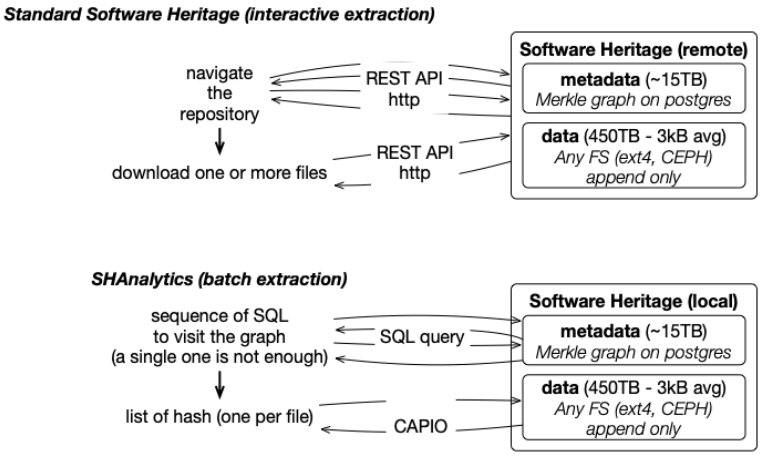


Figure 2.9: SH interactive extraction.

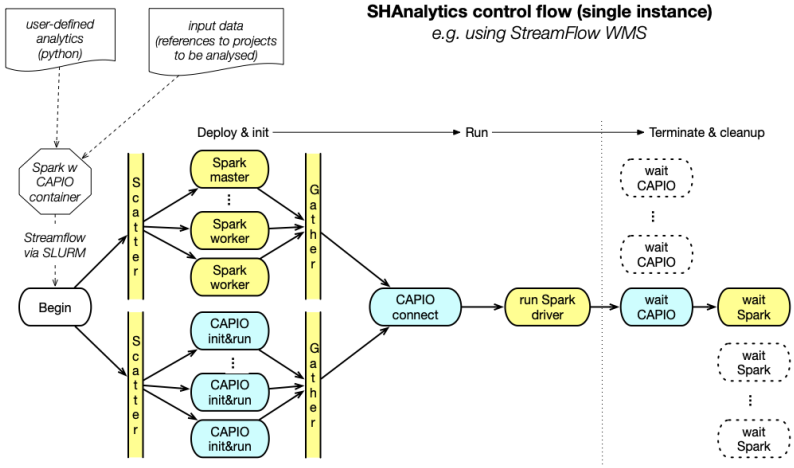


Figure 2.10: A6 Application workflow.

as a continuous data stream of files. These text files are source-code files collected from possibly all reachable public GITs worldwide (currently 400+TB of source files). For its popularity, the streaming framework adopted for the platform is Spark Streaming [1] which provides Java, Scala, and Python bindings to the analytics designers.

Each instance of the Spark Streaming computes one data analysis running within one or more *Singularity* containers, making it possible to deploy them without an installation process. Each Spark instance is deployed either in the single node or multiple node configurations through the StreamFlow [1] workflow management system that interacts with SLURM for resource reservation and actual launching. Spark Streaming, which is typically provided as a (managed) cloud service, is run here via SLURM; similar to an HPC application.

Multiple independent instances of SHAnalytics workflows, each possibly running on multiple nodes, can run at the same time. Each instance executes an ephemeral instance of the Spark framework running a specific analysis. The Spark framework is started by StreamFlow before the analysis and terminates after the completion of the analysis. In principle, the Spark Streaming framework can be substituted with any other data analytics framework that can be launched within containers without affecting the generality of the approach: Flink, PyTorch, R studio, etc.

The SHAnalytics architecture leverages CAPIO (Cross-Applications Programmable I/O) components to

[1] Spark Streaming: https://spark.apache.org/streaming/

read files from SH and expose them as a continuous stream of data to the Spark Streaming applications. CAPIO components also implement a cross-application system-wide file cache of a predefined size that stores the most recent files extracted from the SH repository. The shared files cache is implemented onto a (persistent) distributed file system (NFS, Lustre, etc.) supporting POSIX APIs. The files cache exploits the potential temporal and spatial locality in accessing the same files from distinct data analytics applications to enhance applications' performance.

Concerning the SHAnalytics pipeline, the framework is composed of different modules interacting via explicit messages.The data analytics applications running within the Spark Streaming component follow a data-flow programming model.

The application workload consists of Stream analytics applications which are modeled as Directed Acyclic Graphs (DAG) of operators exchanging data items, often called tuples. Operators process tuples and produce outputs based on their business logic code. Some operators, the stateless ones, may be replicated to increase application throughput. In the most general case, stream analytics applications, when executed on a single scale-up server, are mainly memory-bound (c.f. Word Count, Grep). In the SHAnalytics framework, tuples are text files extracted from the SH repository. In the DAG representing the analytics application, some operators may be more costly in terms of computation time and memory bandwidth requirements than others.

Figure 2.10 shows the SHAnalytics application workflow. For completeness, in the next section, we provide a brief description of the Software Heritage project on top of which SHAnalytics is built.

Software Heritage

Software Heritage is a software stack that enables an API to store and retrieve software repositories worldwide. The mission of SH is to preserve all the open-source code ever produced by humans. It collects and preserves software in source code form to avoid the risk of losing it. SH project is also an extraordinary opportunity for developing new applications performing data analytics on such an enormous corpus of source code to gain useful insights into different fields of computer science, such for example compiler optimizations, software engineering, and machine learning.

Software source code is collected by crawling source code hosting platforms such as GitHub, GitLab, Bitbucket, Sourceforge, and package archives, like npm or PyPI, are ingested and stored in archive (file system) as gzipped files, whose name is a hash that is indexed in a Merkle DAG, which keeps track of identical files across different projects, Gits and file versions. Each artifact in the archive is associated with an identifier called SWHID. An overview of the general architecture of the SH stack is sketched in Figure 2.11.

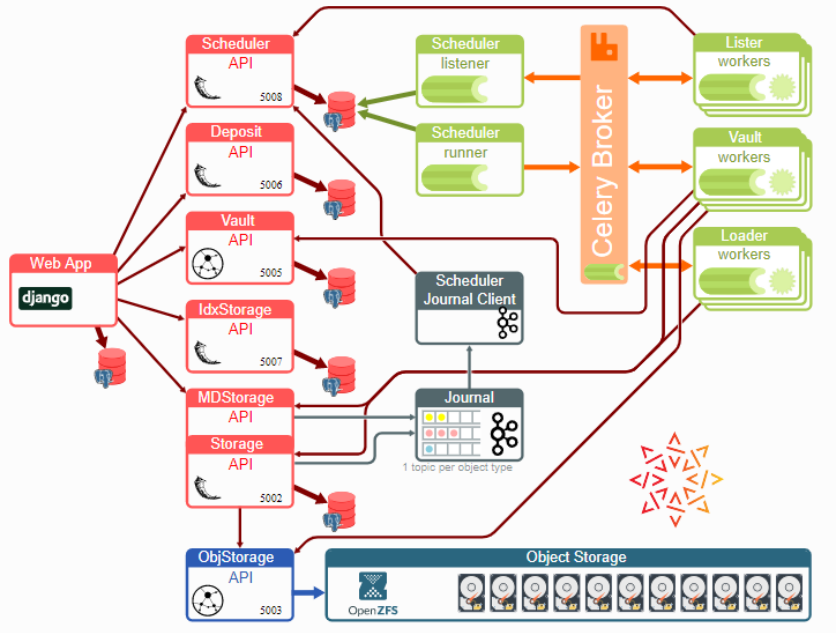


Figure 2.11: SH Overview. Retrieved from the website.

Chapter 3

Application Characterization and Requirements

3.1 Overview

This chapter gives the motivation for collecting specific parameters and their relevance to the ADMIRE project. The information is segregated into different functional groups depending on its characteristics and then gathered in a single template. The skeleton of the template is available under Appendix B.

A general overview of the applications is given in Section 3.2. Apart from general information, it is also necessary to collect I/O relevant data that constitutes the baseline information of the applications, which can potentially support the technologies provided by ADMIRE. The main focus of the requirement collection is to obtain precise data such as the number and size of read/written files, I/O API, etc. This information is presented in Section 3.3.

Following the characterization of the applications, in Section 3.4, a data retention analysis was performed. This study aims to identify the requirements that an I/O subsystem of an HPC system needs to meet. Finally, an analysis of the possibilities of applying the ADMIRE technologies to each of the applications is performed and presented in Section 3.5.

3.2 Application description

The description of the applications is useful for defining the APIs and instrumentation functions to be designed and applied in the context of the ADMIRE project, since these tools must support applications of different fields. In particular, information regarding the programming languages and library dependencies are useful for this purpose. Programming models and communication strategies are also captured, which are needed to examine the malleability possibilities and requirements of the applications. The information collected for all the applications is summarized in Table 3.1 and Table 3.2.

In addition to the collected information, it is also relevant to expose where the applications run for the co-design process in the ADMIRE project. This allows us to get a better picture of performance metrics or statistics presented in the report, which are subsequently analyzed within the scope of ADMIRE. The information is presented in the following list:

- **A1 - ENV**: HPC environment based on Linux (CentOS).
- **A2 - QE**: Cineca cluster, Marconi100 (IBM AC922 “Whiterspoon”).
- **A3 - NEK**: KTH super computer, Beskow (Cray XC40).
- **A4 - RS**: System based on Linux x86_64, 4 $\times$ NVIDIA A100 Tensor Core
- **A5 - SRRF**: Computing node with x86_64 architecture with Intel Xeon and GPU (Nvidia) at PSNC.
- **A6 - SHA**: Currently one Linux server node with two Intel Xeon Gold 6240L and 755GB of RAM.

Application	Programming language	Lines of Code	Library dependencies	Accelerator programming models
A1 - ENV	C++ 17	5000	MPI, OMP, CUDA	CUDA
A2 - QE	Fortran 90, Fortran2003, C	≈ 48000	MPI, LAPACK, BLAS	-
A3 - NEK	Fortran 77, C	Fortran:70000, C:30000	Fortran and C compilers, MPI	-
A4 - RS	Python	400	Horovod, Tensorflow	CUDA
A5 - SRRF	Java	≈ 2000	Aparapi, OpenCL	OpenCL
A6 - SHA	Java, Scala, Python	-	Spark Streaming framework	-

Table 3.1: Applications description - A.

Application	Task-based programming models	Other multi-threading approaches	Workflow engine	Inter-process communication technique
A1 - ENV	OpenMPI	-	DagOnStar	MPI
A2 - QE	OpenMP	-	-	MPI
A3 - NEK	-	-	-	MPI
A4 - RS	-	-	-	MPI and/or NCCL
A5 - SRRF	-	-	-	-
A6 - SHA	-	-	StreamFlow	Analytics: Spark Streaming ecosystem. CAPIO: MPI and POSIX shared-memory.

Table 3.2: Applications description - B.

3.3 I/O Analysis

The I/O analysis in this report is further subdivided into three parts:

- **I/O behavior**: This part is concerned with gathering data about the general behavior of I/O activity in the system when a given application runs. The aim of collecting this information is to estimate how the system is affected by the application and to provide information to other ADMIRE technologies on how to manage it. The information is gathered in Table 3.3.

- **I/O implementation**: This part aims to understand **how** each of the applications is performing I/O operations. This is useful for the technical work packages to design operations that might optimize performance. Table 3.4 summarizes the information.

- **File characteristics and fault tolerance mechanisms**: This part aims to collect more information on what specific file sizes are present in the execution of the applications, as well as give more insight into the objective of such files, which might allow understanding the permanency of the files in the system. Table 3.5 collects the information.

Application	**Data read-/written**	**Amount of IOPs** ($\frac{sizeofOutput}{sizeofInput}$)	**Fraction of small/large I/O request**	**I/O intensity**	**Access patterns**
A1 - ENV	-/1428GB	> $O(10^2)$	-	-	-
A2 - QE	1MB/114GB	$O(10^5)$	from 100MB to 5.5GB	$\approx 20\%$	Sequential
A3 - NEK	28GB/389GB	$O(10)$	Constant file size - 6.6GB	$\approx 2\%$	Sequential
A4 RS	60GB/500MB	O(100)	100MB	-	Sequential and distributed
A5 - SRRF	600MB/- (per run)	≤ 1	90% large I/O	$\approx 5\%$	Sequential
A6 - SHA	$O(10^2)$GB/-	≤ 1	Many small reads/writes	-	Random

Table 3.3: I/O behavior.

Application	**I/O API**	**Object Storage**	**File format**	**I/O Parallelism**	**Shared access**
A1 - ENV	HDF5	In development	NetCDF data (based on HDF5)	N/A	No
A2 - QE	POSIX I/O	No	Text, binary and xlm	N/A	Yes
A3 - NEK	MPI I/O, POSIX I/O	No	Binary and txt log files	MPI I/O	Yes
A4 - RS	MPI/NCCL, HDF5, TensorFlow Dataset API	No	tfrecord, h5	N/A	Yes
A5 - SRRF	-	No	Images in *.tif format	N/A	-
A6 - SHA	POSIX I/O [1]	No	Text and binary files	Shared cache	Yes

Table 3.4: I/O implementation.

Application	File creations	Max, min, avrg file size	number of files	Fault tolerance mechanism and API
A1 - ENV	1344	2.8 GB, 220 MB, 525 MB	600	Stop/restart
A2 - QE	19	5.5 GB, - , 600MB	30	N/A
A3 - NEK	59	13GB, - , 6.6GB	66	Checkpointing with MPI I/O
A4 - RS	1 per 10 epochs	- , - , 500MB	30	Checkpointing with Tensorflow
A5 - SRRF	1 per run	- , 500MB, 2GB	1 per run	N/A
A6 - SHA	Few outputs	- , - , $\approx$ 3KB	-	N/A

Table 3.5: File characteristics and fault tolerance mechanisms.

3.4 Data retention analysis

Data retention analysis can be used to determine the requirements for the I/O subsystem of an HPC system or, in particular for ADMIRE, the ad-hoc storage system under WP2 supervision. The concept of data retention comes from the fact that during the workflow, the applications will produce or consume data sets that have different lifetimes that can be, in principle, identified and recorded in a diagram. For the analysis of the applications, three main periods for file life have been defined:

- **Transient files**: Data discarded on simulation completion or when later processing steps are concluded.
- **Short term files**: Data used throughout the execution of the scientific workflow.
- **Permanent files**: Data outliving the system used to generate it.

Being able to understand the data behavior for each application would allow to better select how to apply the ADMIRE technologies on them, as transient data sets typically benefit from fast storage rather than capacity, while the latter is essential for systems with applications creating many permanent data sets. The analysis in this section is difficult to perform since good knowledge of the application, system, and usage is required, thus only the applications with a complete analysis will be shown below while those lacking the complete information will be pending for future updates.

For the sake of convenience, the diagrams have been compiled in Appendix C while general remarks on the information contained for each of the applications are presented below.

1. **A1 - ENV**: The data retention of the environment application was performed for all the current models that it supports. The diagrams can be seen in Figures C.1 to C.3. They show how at the initial stages in the pipeline, more transient and short-term files are present, while the data gets a more permanent nature as the final results are obtained.
2. **A2 - QE**: Data retention diagram for QE Car-Parrinello molecular dynamics simulation is shown in Figure C.4. After the simulation is set up by reading input parameters and pseudopotentials, it runs for n steps. Depending on specification of the input parameters, restart files and output files can be printed

[1]Data analytics applications do not read files using POSIX I/O, instead they are provided in input as a stream. POSIX I/O operations are used by the CAPIO component.

every m steps out of n with $0 \leq m \leq n$. The dashed lines indicate that files may be permanent or not depending on the value of m.

3. **A3 - NEK**: Data retention of Nek5000 is the standard for simulation applications. Files at given time steps are written to the file system and are stored permanently for post-processing, while restart files or checkpoints are only transient files needed to complete the work cycles. Figure C.5 shows the data retention for different stages of the pipeline.

4. **A4 - RS**: The data retention diagram for the remote sensing application can be observed in Figure C.6. Most of the data is permanent, as there are no significant intermediate I/O operations other than checkpointing for fault tolerance.

5. **A5 - SRRF**: The data retention of the application can be observed in Figure C.7. For this application, all the data (images) are read from the file system and processed by the GPUs. The outputs are then stored permanently without any sort of intermediate storage step.

6. **A6 - SHA**: In the diagram, the use of dashed lines indicates that the component may be optional depending on the analytics application executed. The data retention diagram for SHA can be seen in Figure C.8, where an array of transient and short-term data is visible which could benefit from the emerging technologies proposed in ADMIRE.

From the analysis, it is possible to observe that even though the outputs of the applications tend to be permanent, as is expected, many of them produce intermediate data that engages the I/O system but does not need to be stored permanently. This sort of data can open avenues for performance improvement leveraging technologies in the ADMIRE project such as the ad-hoc storage system.

3.5 ADMIRE technologies applicability

This section aims to get a picture of which applications can be used to showcase specific ADMIRE technologies. The answers in the section contain insights into the applications rather than detailed descriptions on how technologies can be implemented as this is out of the scope of the document. Table 3.6 summarizes the information, where an "X" symbolizes that the application is compatible and could benefit from the specific ADMIRE technology while "TBD" means that further analysis must be done to determine the feasibility of the technology in the application. Specific ways in how each application might interact with the technology can be found in the application requirement templates.

Application	**Ad-hoc storage system**	**Malleability management and control**	**I/O Scheduling**	**In-situ operations**
A1 - ENV	X	X	X	X
A2 - QE	X	TBD	-	-
A3 - NEK	X	X	X	X
A4 - RS	X	TBD	X	TBD
A5 - SRRF	TBD	TBD	X	TBD
A6 - SHA	X	-	X	X

Table 3.6: ADMIRE technologies applicability.

As a general note, the application experts coincide in the fact that the applications can be instrumented and slight coding changes are acceptable. However, the implementation of technologies that would imply a redesign

of sections of the applications is generally not acceptable. Nevertheless, we consider this to be an acceptable restriction to the technical sections of the project as it is expected that once the project is completed, it is most likely that it will be used by a wider audience if changes to the applications are kept minimal.

For completeness, a brief explanation of why some applications cannot, at present, be used for a specific technology is given in the following list:

- Malleability management in A6 - SHA depends on the malleability features offered by the Spark Streaming framework. At the current status of the framework, is difficult to foresee that ADMIRE's malleability features might be applied.
- For the I/O scheduling, the applications where the ADMIRE technologies are not applicable share the fact that it is not possible to know the I/O requirements ahead of time, thus, I/O hints to the scheduler are not possible. For these cases, ADMIRE can explore if the scheduler can rely on previous simulations or job history in order to get the information needed.
- Applications that can not benefit from in-situ transformations share the main characteristic that the data will be used for post-processing approaches that need high accuracy or the fact that the most effective way of processing the data has been achieved.

Chapter 4

Requirements Summary, Insights and Future Work

The applications' I/O requirements were stated in Chapter 3, however, it is possible to point out common characteristics or trends between applications. The purpose of this chapter is to document these commonalities. Each of the following sections references one of the subsections treated in Chapter 3.

4.1 Application description

From the application descriptions, it is possible to observe that the ADMIRE technologies must be compatible with a variety of systems and applications written in different programming languages. They must also support operations in accelerators such as GPUs and a distinct number of file formats. Workflow engines are also used, thus, the ADMIRE technologies must be compatible with these.

4.2 I/O Analysis

In this category, the following points are worth further discussion:

- From the I/O characterization, it is possible to observe that most applications have a relevant I/O activity, with read or written data in the order of hundreds of GBs while there is a balanced number of write-heavy and read-heavy applications. As a preliminary judgment, read-heavy applications should be considered for pre-loading operations, while those that have a large amount of output data could be the target of malleability management to improve the system capabilities.
- The I/O intensities between the application vary, and in some cases the output frequency can be set by the user, thus higher or lower intensities can be manifested in production runs not associated with the profiling purpose of this document. The ADMIRE technologies must be designed setting tolerances to keep this in mind.
- The fact that the applications have different fractions of large vs small I/O requests is ideal for the evaluation of technological impacts of the ADMIRE technologies on the system.
- The I/O access patterns are mostly sequential, which can give insights on how the ad-hoc storage tiers should be designed.
- Although object storage capabilities would be ideal, it is clear that support for this technology is not needed at the initial stages, as only one application can take advantage of this technology at present.
- There is a level of I/O parallelism in some applications, which is beneficial for performance but it means that the system must provide a large bandwidth such that the operations can be executed at peak performance.

4.3 Data retention

The data retention diagrams present in this document provide valuable insights on each of the applications' I/O interactions. They offer some tools for ADMIRE's technical work packages in order to understand and design strategies while considering the necessities of the applications. Even though the information has already been provided, some insights from the results are summarized below.

The data flow patterns and data retention of the applications within the ADMIRE project vary but they can ultimately be seen in terms of groups. From the figures in Section 3.4 and information captured regarding the data flow of the applications, it is possible to identify that while many of them perform periodic I/O requests, they are not always done for the same purpose, which in turn changes the nature of how the data streams can be dealt with. For example, applications such as **Remote Sensing** produce periodic data almost exclusively in the form of checkpoints, which are used as a form of fault tolerance mechanism but are not the outputs of the application. This means that a certain type of flexibility is present in this sort of operation, where the application could delay or advance the creation of checkpoints depending on the system I/O state.

This sort of operation is a significant challenge while running simulation applications such as **Nek5000**, **Environmental application**, and **Quantum Espresso** where the periodic output data sets are the outputs of the application. For these cases, data is produced at different states (e.g., time steps) of the physical system that they are simulating, and such data is relevant for the analysis or post-processing of the results. In a case like this, periodic outputs are usually required at specifics steps or iterations and thu must not be delayed. *Therefore, it is important that these limitations are taken into consideration in the implementation of I/O system relieving strategies within the ADMIRE project.*

Notably, the simulation applications tend to write files to I/O frequently, but the amount of data that they read is limited to set up files and initial/boundary conditions at the start of every pipeline step. Machine learning applications such as **SRRF** and the service **SH** possess a larger amount of data that must be read and, even though the most significant effort is at the start of the execution as well, the amount of data is orders of magnitude higher than for other applications. Further pre-loading capabilities should be investigated to help these applications speed up their executions, which is in line with the ad-hoc storage system proposed by ADMIRE.

4.4 ADMIRE technologies applicability and Malleability

From Table 3.6, it is possible to observe that for all of the admire technologies, a large portion of the applications can be instrumented. Explanations on why some of the technologies do not apply to particular codes were also given in Section 3.5, however some extra remarks can be made.

Control and malleability are key aspects of the ADMIRE project, yet in Table 3.6 some of the applications express that further investigation needs to be done in order to correctly asses the applicability of the technology. The reason for this is that the investigation done in this deliverable refers mostly to malleability with respect to computational resources (e.g. number of nodes used, etc.) opposed to malleability in bandwidth where, for most cases, only positive effects can be obtained. In the applications descriptions in Chapter 2, several of the applications show their performance dependence on the computational resources they acquire. In the applications description chapter it is possible to observe that, as expected, performance usually increases with increased resource allocation. While it is generally accepted that malleability would have positive effects on the performance for all applications. It is in the implementation of the technology where most of the applications express their doubts. This should be regarded as an understandable behaviour as most of the applications in the co-design process of the ADMIRE project were not designed with malleability in mind. Having stated this, each of the applications will closely work with the technical work packages in order to address the main concern, i.e, how the redistribution of work when new processes are added to the application is done. Having a clear picture of this, the scope of code changes that might be needed in order to apply the technology would be discussed and the full advantages of malleability will be harvested if applicable. Always keeping in mind that modifications to the applications need to be kept to a minimum to ensure that more applications adopt the ADMIRE technologies once the project is done. It is noted that the same concerns would likely be present in new applications trying to adopt ADMIRE in the future, and having close discussions with the technical work

packages would strengthen the consortium in their response to such doubts.

4.5 Requirements

Based on the characterization of Chapter 3 and the previous discussion in the present chapter, the following list of requirements has been obtained. It aims to summarize the application requirements in a more general way.

- Systems with Unix-based environments must be supported.
- Accelerators such as GPU must be supported.
- The technologies must guarantee system performance for applications with up to $O(10^1)$ TB of write operations per run.
- The technologies must guarantee system performance for applications with up to $O(10^2)$ GB of read operations per run.
- The technologies must guarantee system performance over a large range of I/O request sizes. From $O(1)KB$ to $O(10)GB$
- The performance must be improved for applications with sequential and random access patterns.
- A large array of I/O API must be supported. See Table 3.4.
- Object storage systems are not necessary at the initial stages.
- An array of file formats must be supported. Some applications use custom formats that need to be taken into consideration for any in-situ processing of the data.
- High bandwidth must be guaranteed for improved performance of applications with I/O parallelism.
- The technologies must improve performance for applications with a large number of file creations up to the order of thousands of files per run.
- Any transient storage system must possess enough capacity to temporarily hold files of up to $O(10)$GB at a given time.
- Checkpoint and periodic output performance must be increased.
- Rather small code changes are allowed for the instrumentation of most of the applications.
- The instrumentation tools must support a variety of programming languages. See Table 3.1.
- The technical work packages must have individual discussions with the application experts to determine how malleability management can be introduced. It is not obvious that the number of processes can be dynamically modified for several applications.

4.6 Conclusion

The ADMIRE project supports a broad range of data-intensive applications which produce large amounts of data in current systems. Application characterization and requirement elicitation have been performed by systematically analyzing the main objectives of the project, the Key Performance Indicators of the proposal, and the aims of the ADMIRE technologies. Preliminary analysis on where to use the technologies has been performed, which will serve as a baseline for other ADMIRE technical work packages to commence their API designs. Nevertheless, a further detailed investigation must be performed by the application experts to identify the optimal way of implementation(s).

This document aims to serve as a baseline of the application requirements, and the information and methodology hereby contained will be used for future stages of the co-design process.

Appendix A

Annex I: Acknowledgements

This document has been compiled thanks (but not limited) to the effort of the application representatives in work package 7:

- Rocco Sedona (JSC)
- Surbhi Sharma (JSC)
- Gabriele Cavallaro (JSC)
- Andrea Piserchia (CINECA)
- Massimo Torquati (CINI - UNIPI)
- Alberto R. Martinelli (CINI - UNITO)
- Barbara Cantalupo (CINI - UNITO)
- Marco Aldinucci (CINI - UNITO)
- Viviana Bono (CINI - UNITO)
- Raffaele Montella (CINI - UNIPARTHENOPE)
- Diana Di Luccio (CINI - UNIPARTHENOPE)
- Adalberto Perez (KTH)
- Stefano Markidis (KTH)
- Daniele Gregori (E4)
- Maciej Figiel (PSNC / IBCh)
- Piotr Piasecki (PSNC / IBCh)

Appendix B

Annex II

ADMIRE WP7 Application – requirements and characteristics template.

I suggest we use this template separately for each use case (one form per use case)

Application title: Place Holder.

Scientific domain:

Publication/website:

1. General description of the application

Please provide a general description of the application with the following information.

a. *General description of the application, accounting for scientific context.*
b. *Application workflow.*
c. *How do executables interact within the scope of your application (if more than one)?*
d. *Which executables perform the I/O intensive operations?*

1.1 Estimated workload

Please describe the estimated workload for a reference case that you consider relevant for the ADMIRE project. You can follow the next guidelines and add more information that you consider important.

a. *Estimated workload.*
b. *Is there a dominant kernel in the work performed? e.g., matrix-matrix multiplications, etc.*
c. *What is the impact of I/O in the application? (approx % of I/O compared to execution time)*

1.2 Environment, code characteristics and libraries, third-party dependencies

Describe the application environment. You can use the following guidelines:

a. *State the hardware platform the application currently runs on (architecture, OS)*
b. *What are the application's third party dependencies and libraries?*
c. *How does the application launch on the compute nodes (e.g. mpirun, SLURM, etc.)*
d. *If the binaries are invoked via a script, can you provide an example snippet of the script?*

1.3 Code description, programming models, workflow technologies

Please fill in the following tables and add any information that you consider relevant.

Code description

Programming language(s)	
Total number of lines of code (ballpark figure)	
Library dependencies	
Is mini/toy-application available?	

Programming models

Accelerator programming models (e.g. CUDA, OpenCL, OpenACC, OpenMP)	
Task-based programming models (e.g. OpenMP, OpenSs, Intel TLB)	
Other multi-threading approaches (e.g. POSIX threads)	

Workflow technologies

Workflow engine (e.g. Kepler, Pegasus, Airflow)	
Inter-process or inter-actor communication techniques (e.g. Hadoop, MPI, Flink)	

Additional application characteristics

Add information about the application that you consider relevant, but that was not included before.

1.4 Parallel performance and optimization.

Allow us to get a picture of a typical performance.

a. *For which particular architecture is the application optimized for? Does it support CPU, GPU, or other accelerators?*

2. I/O characterization

The aim of this section is to collect the application requirements. This section provides baseline information of the applications which might serve as restrictions for other WPs and can be used at later stages to measure the effectiveness of the ADMIRE technologies.

2.1 I/O interfaces, libraries, file formats

Please, fill in the information and add additional information you consider relevant.

a. *What type of I/O interface is used?. for example MPI I/O, HDF5, … . Is I/O handled by libraries, if so, which?*
b. *What file formats are used by the application? (they could be different for different phases: development, debugging and production)*
c. *Does the application support object storage?DDN WOS, intel DAOS, DataClay etc.*

2.2 Data flow

Please describe the amount of data moved from and to the I/O system for a reference use case relevant for the ADMIRE project.

2.3 Characterization of I/O patterns

Please fill in the following tables and add any extra relevant comments/requirements. Provide the information for a reference case that you consider relevant for the ADMIRE project.

To obtain the numerical information, you can use a profiler such as TAU (https://github.com/UO-OACISS/tau2), Darshan (https://github.com/darshan-hpc/darshan) or any other profiling tool you have access to.

I/O Characteristics and Patterns.

Total amount (size) of data read/written	
Total amount of IOPs (input data vs generated output data)	
Amount of unique data read (how much data is re-read from data source)	
I/O intensity: time spent in I/O vs. total execution time (at what rate?)	
Fraction of small and large I/O requests.	
Access patterns dominantly sequential or random	
I/O API (e.g MPI I/O, POSIX I/O, HDF5, etc)	
I/O parallelism (I/O from single or multiple processes)	
If using parallel I/O, is file/object access shared?	
For shared access, what kind of inter-process I/O consistency is	

assumed? (commit-on-close, POSIX semantics, etc.)?	

File characteristics.

Total number of file/object creations (output files)	
Maximum, minimum and average file/object size	
Maximum number of files in a particular directory	

Fault tolerance.

Supported fault-tolerance mechanisms (checkpointing, transactions, etc.) and what APIs are used?	
Fault-tolerance related mechanisms needed from the underlying system (fsync, ACID transactions, HA, replication etc.)?	

I/O Behaviour

a. *How is the input data managed by the application? Is it replicated or distributed? If distributed, each processor or only one processor reads and distributes the data (broadcast when MPI)?*
b. *What tools, either conventional diff or specialized, you use to debug the application's IO behavior? For instance to validate outputs.*
c. *Is your application using intermediate or primary storage mechanisms to store intermediate results, e.g. burst buffers or similar mechanisms? Explain how and in what circumstances? Note: This could be used in DAG-based multi-job computations: job chaining as an example.*
d. *If there is any checkpointing being implied to mitigate data loss, how is the checkpointing implemented? What does the data structure look like? Or the application is checkpointing use underlying tools, which the application developer is not aware of? What is the checkpoint foot print in terms of storage consumption? What is the checkpoint read and write overhead and which mode is dominating, experience-wise?*

2.4 Challenging I/O aspects for the applications.

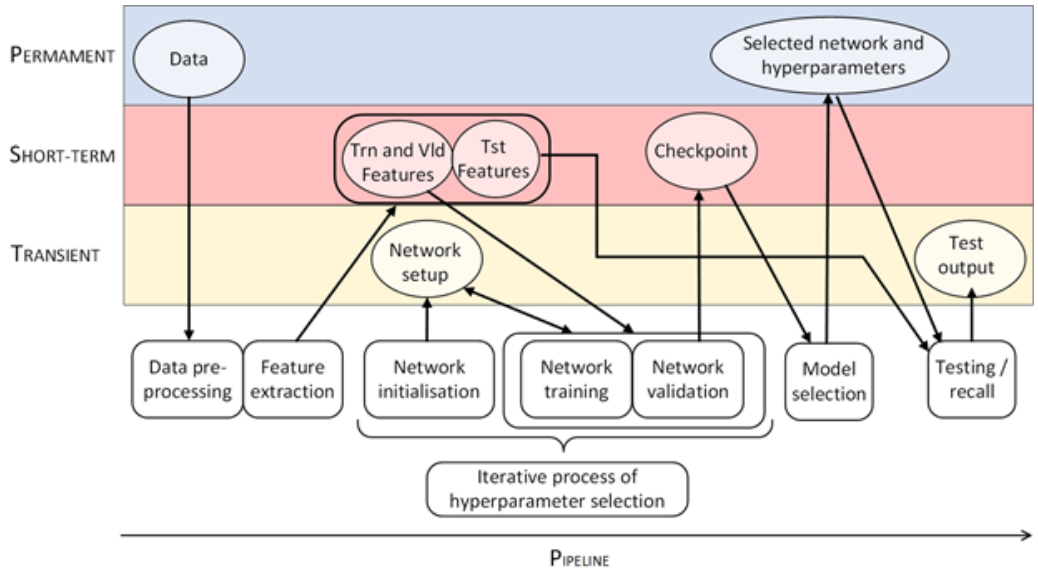
Please characterize key bottlenecks for scalability as well as storage and data management.

a. *What are the key bottlenecks for scalability in the application? What are the key bottlenecks for storage and data management in the application?*

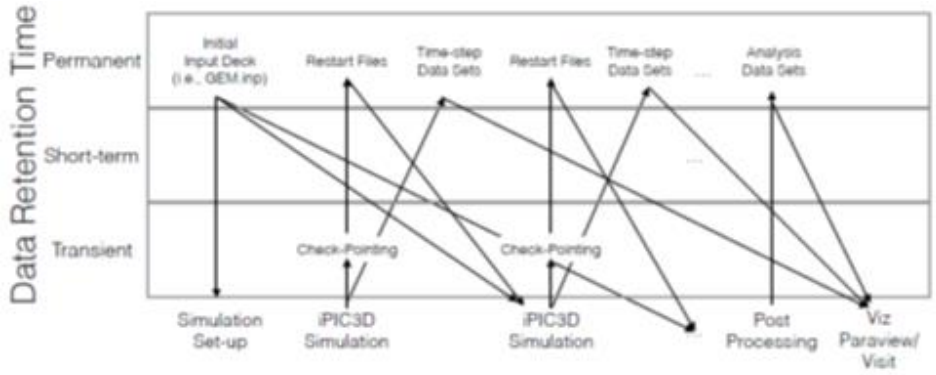
2.5 Data retention diagram.

Please construct a data retention diagram. It is a diagram that allows us to identify the type of data used by the application at different stages of the pipeline. Here we show 2 examples that can be adapted to your application. Note that these diagrams include all phases and not only the execution phase.

Example for a machine learning application:



Example for a simulation application:



3. Applications in the context of ADMIRE.

The aim of this section is to get a picture of which applications can be used to showcase specific ADMIRE technologies. The answers in the section should contain insights into the applications rather than detailed descriptions on how technologies can be implemented (out of the scope of the document).

3.1 Work Package 2 deals with an ad-hoc storage system. The main idea is to efficiently use node internal NVMe and persistent memory technologies to reduce the pressure on the back-end storage system. In this context:

a. *Does the application provide opportunities for performing independent I/O by pre-loading in-out data and performing asynchronous I/O*

3.2 WP 3 will develop mechanisms to manage the combined malleability of computation and I/O and WP 6 will provide an intelligent controller to dynamically steer the systems components. in this context:

a. *The applications will likely need to be instrumented with the libraries or API produced by the technical work packages. Please provide insight on what type of changes might be acceptable for the main branch of the code.*
b. *For control and malleability, control points in the application need to be specified. Please provide insight on what sections of the application might benefit or be instrumental for malleability.*
c. *As a result of control policies, applications might be reconfigured. i.e., the number of processes running the application might be dynamically changed in a temporal or permanent manner in order to increase system performance. Please provide insight in regards to the flexibility of the application for such tasks.*

3.3 WP 4 will provide inter-job storage scheduling and I/O coordination. In this context:

a. *Is it possible to know the (unprofiled) application I/O requirements ahead of time such that a user could provide batch system hints while requesting an allocation? e.g which data will be accessed by the batch job, the type of access, the expected lifetime, expected reuse profile.*
b. *Can the application be benefited by in-situ/in-transit data transformations such that the back-end storage system is off loaded. (by means of data compression, in-situ visualization, etc.)*

3.4 Other information, extra requirements

*Here you could provide any other relevant comments and, importantly, specify your **wish list** of the requirements that have not been accounted for earlier in the questionnaire – opportunities and problems foreseen in the context of the platform usage.*

Appendix C

Annex III: Data retention diagrams

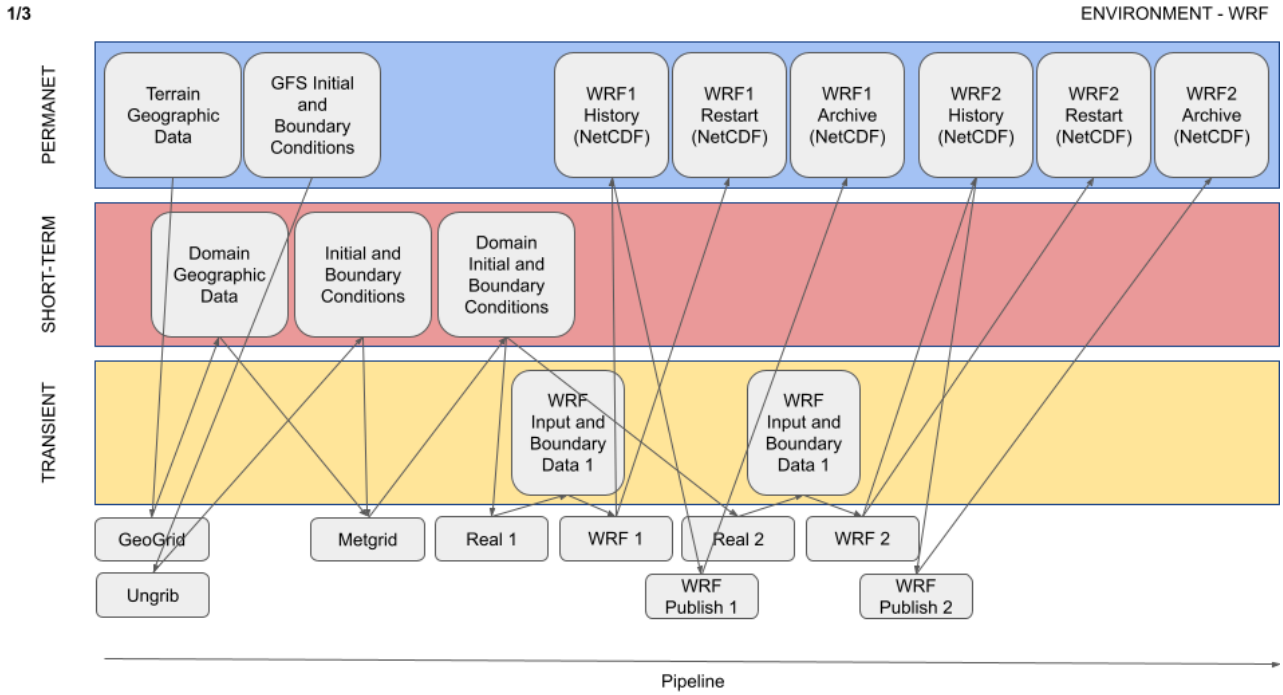


Figure C.1: Data retention Environmental application - 1/3.

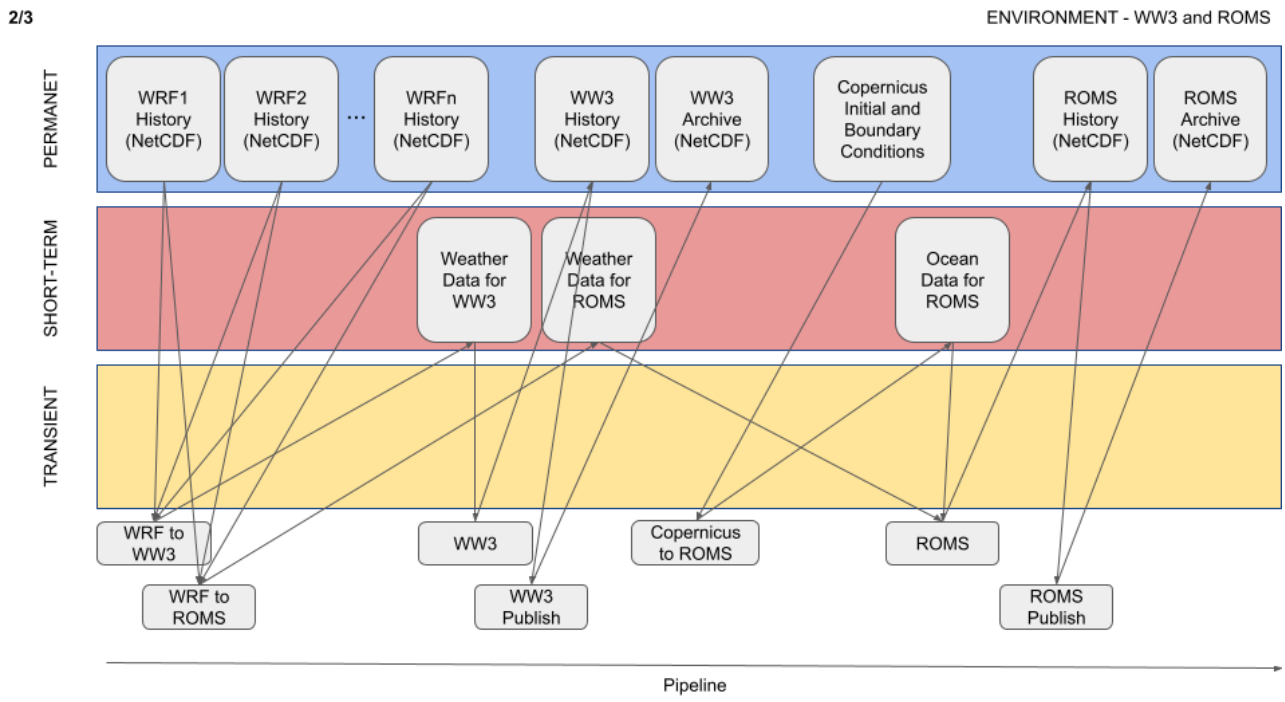


Figure C.2: Data retention Environmental application - 2/3.

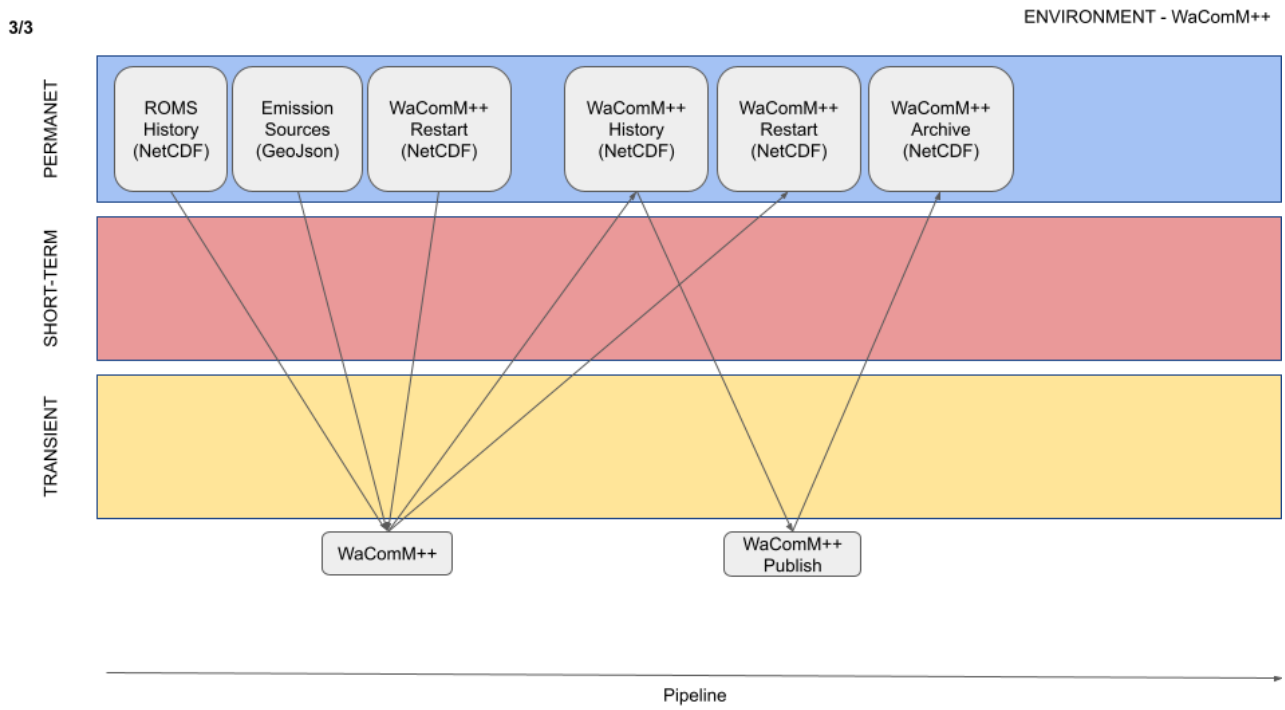


Figure C.3: Data retention Environmental application - 3/3.

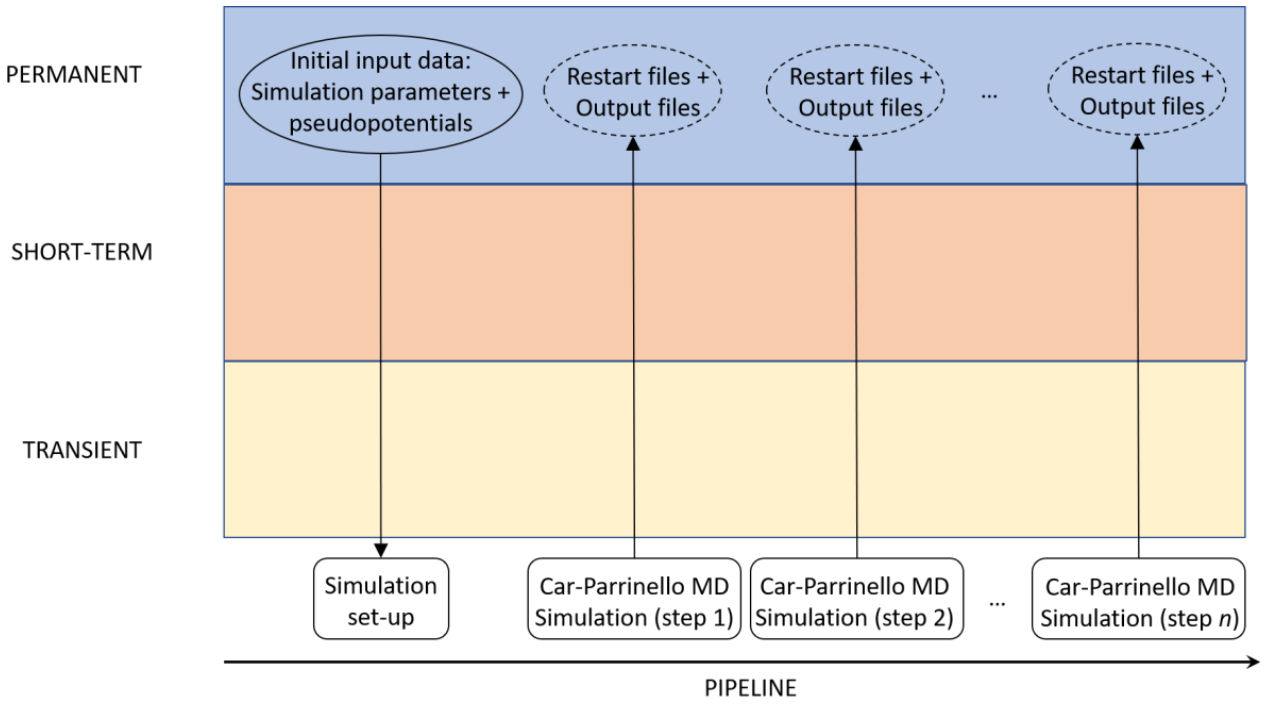


Figure C.4: Data retention for QE.

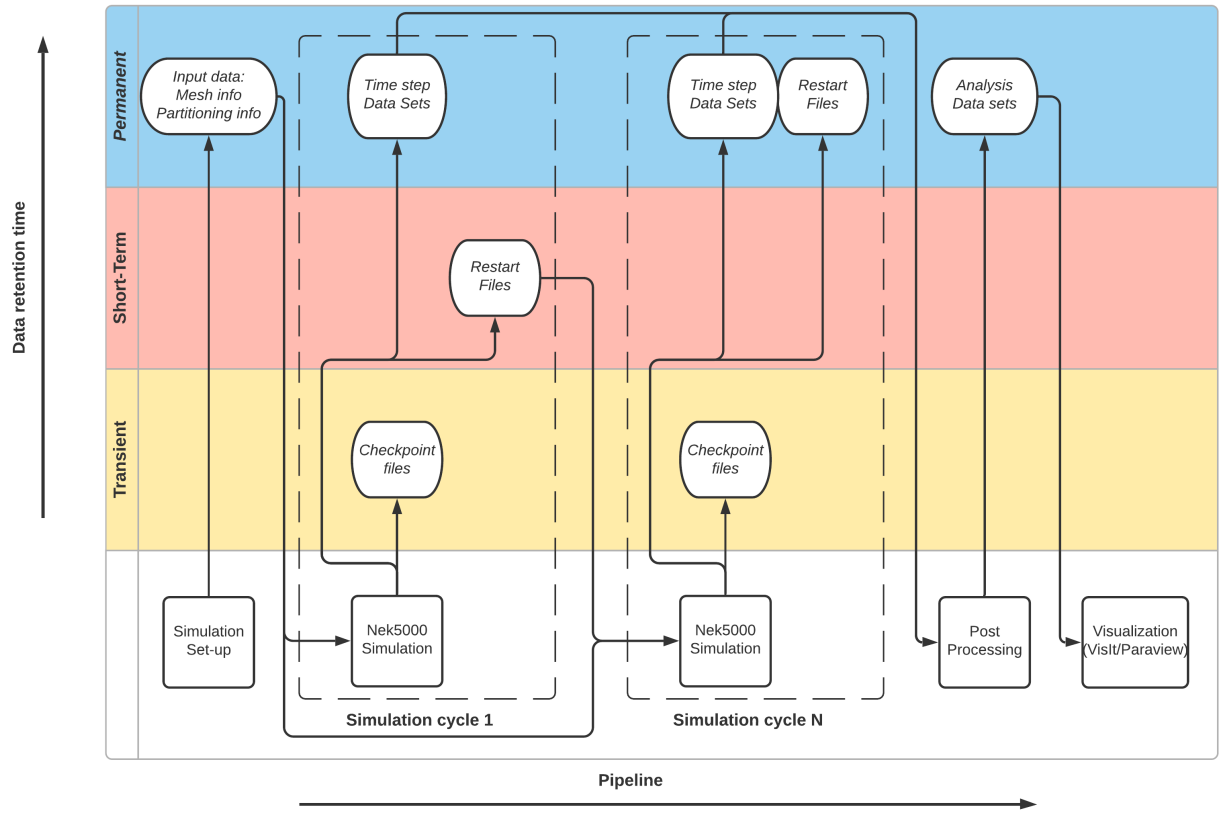


Figure C.5: Data retention for Nek5000.

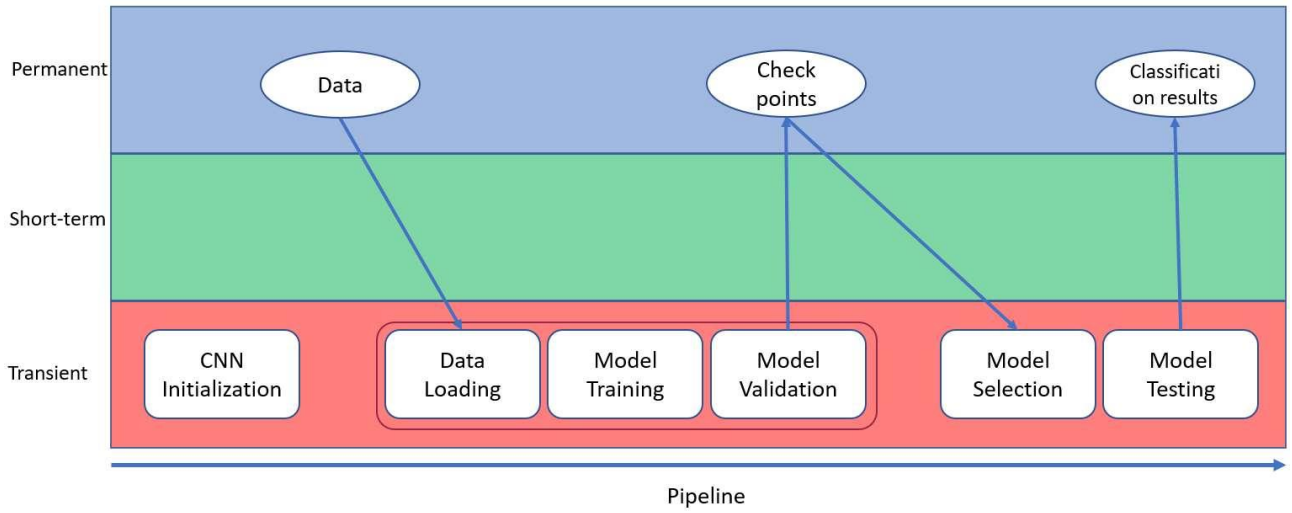


Figure C.6: Data retention for Remote sensing.

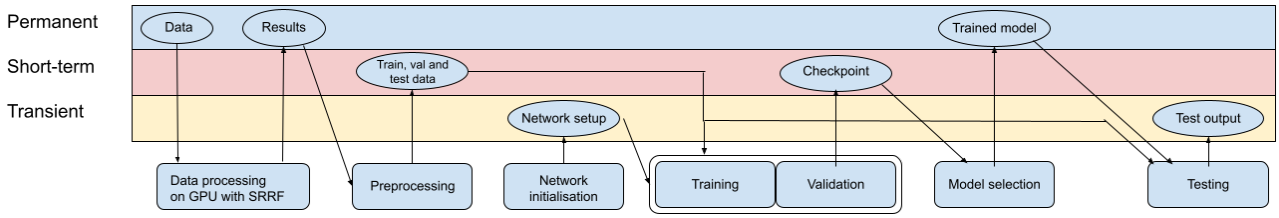


Figure C.7: Data retention for SRRF.

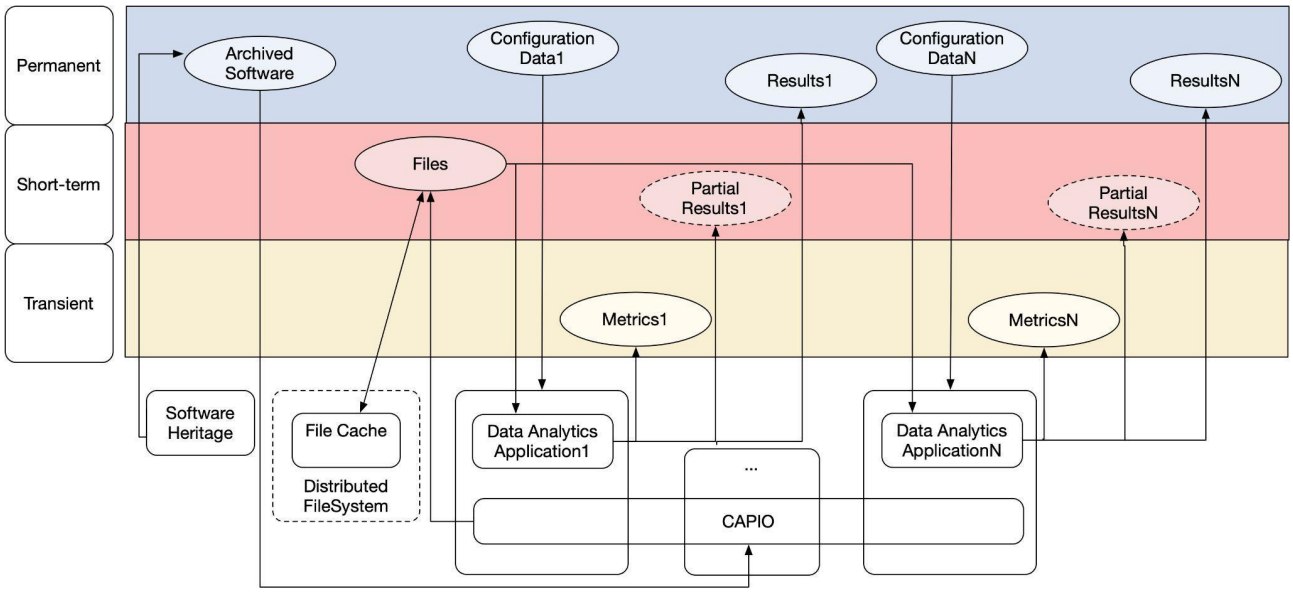


Figure C.8: Data retention for SHA.

Bibliography

[1] Iacopo Colonnelli, Barbara Cantalupo, Ivan Merelli, and Marco Aldinucci. Streamflow: cross-breeding cloud with HPC. *IEEE Transactions on Emerging Topics in Computing*, 2020.

[2] Patrizia De Gaetano, Andrea M Doglioli, Marcello G Magaldi, Paolo Vassallo, and Mauro Fabiano. Foam, a new simple benthic degradative module for the lamp3d model: an application to a mediterranean fish farm. *Aquaculture Research*, 39(11):1229–1242, 2008.

[3] Diana Di Luccio, Ardelio Galletti, Livia Marcellino, Angelo Riccio, Raffaele Montella, and Alison Brizius. Some remarks about a community open source lagrangian pollutant transport and dispersion model. *Procedia computer science*, 113:490–495, 2017.

[4] Diana Di Luccio, Angelo Riccio, Ardelio Galletti, Giuliano Laccetti, Marco Lapegna, Livia Marcellino, Sokol Kosta, and Raffaele Montella. Coastal marine data crowdsourcing using the internet of floating things: Improving the results of a water quality model. *IEEE Access*, 8:101209–101223, 2020.

[5] AM Doglioli, MG Magaldi, L Vezzulli, and S Tucci. Development of a numerical model to study the dispersion of wastes coming from a marine fish farm in the ligurian sea (western mediterranean). *Aquaculture*, 231(1-4):215–235, 2004.

[6] P. F. Fischer, J. W. Lottes, and S. G. Kerkemeier. nek5000 Web page: http://nek5000.mcs.anl.gov, 2008.

[7] P Giannozzi, O Andreussi, T Brumme, O Bunau, M Buongiorno Nardelli, M Calandra, R Car, C Cavazzoni, D Ceresoli, M Cococcioni, N Colonna, I Carnimeo, A Dal Corso, S de Gironcoli, P Delugas, R A DiStasio Jr, A Ferretti, A Floris, G Fratesi, G Fugallo, R Gebauer, U Gerstmann, F Giustino, T Gorni, J Jia, M Kawamura, H-Y Ko, A Kokalj, E Küçükbenli, M Lazzeri, M Marsili, N Marzari, F Mauri, N L Nguyen, H-V Nguyen, A Otero de-la Roza, L Paulatto, S Poncé, D Rocca, R Sabatini, B Santra, M Schlipf, A P Seitsonen, A Smogunov, I Timrov, T Thonhauser, P Umari, N Vast, X Wu, and S Baroni. Advanced capabilities for materials modelling with quantum espresso. *Journal of Physics: Condensed Matter*, 29(46):465901, 2017.

[8] Paolo Giannozzi, Stefano Baroni, Nicola Bonini, Matteo Calandra, Roberto Car, Carlo Cavazzoni, Davide Ceresoli, Guido L Chiarotti, Matteo Cococcioni, Ismaila Dabo, Andrea Dal Corso, Stefano de Gironcoli, Stefano Fabris, Guido Fratesi, Ralph Gebauer, Uwe Gerstmann, Christos Gougoussis, Anton Kokalj, Michele Lazzeri, Layla Martin-Samos, Nicola Marzari, Francesco Mauri, Riccardo Mazzarello, Stefano Paolini, Alfredo Pasquarello, Lorenzo Paulatto, Carlo Sbraccia, Sandro Scandolo, Gabriele Sclauzero, Ari P Seitsonen, Alexander Smogunov, Paolo Umari, and Renata M Wentzcovitch. Quantum espresso: a modular and open-source software project for quantum simulations of materials. *Journal of Physics: Condensed Matter*, 21(39):395502 (19pp), 2009.

[9] Paolo Giannozzi, Oscar Baseggio, Pietro Bonfà, Davide Brunato, Roberto Car, Ivan Carnimeo, Carlo Cavazzoni, Stefano de Gironcoli, Pietro Delugas, Fabrizio Ferrari Ruffino, Andrea Ferretti, Nicola Marzari, Iurii Timrov, Andrea Urru, and Stefano Baroni. Quantum espresso toward the exascale. *The Journal of Chemical Physics*, 152(15):154105, 2020.

[10] G Giunta, R Montella, P Mariani, and A Riccio. Modeling and computational issues for air/water quality problems. *A grid computing approach. Il Nuovo Cimento C*, 28, 2005.

[11] N. Gustafsson, S. Culley, Ashdown G., and et al. Fast live-cell conventional fluorophore nanoscopy with ImageJ through super-resolution radial fluctuations. *Nature Commun*, 7(12471), 2016.

[12] L. Hufnagel, J. Canton, R. Örlü, O. Marin, E. Merzari, and P. Schlatter. The three-dimensional structure of swirl-switching in bent pipe flow. *Journal of Fluid Mechanics*, 835:86–101, 2018.

[13] Raffaele Montella, Alison Brizius, Diana Di Luccio, Cheryl Porter, Joshua Elliot, Ravi Madduri, David Kelly, Angelo Riccio, and Ian Foster. Using the face-it portal and workflow engine for operational food quality prediction and assessment: An application to mussel farms monitoring in the bay of napoli, italy. *Future Generation Computer Systems*, 110:453–467, 2020.

[14] Raffaele Montella, Diana Di Luccio, and Sokol Kosta. Dagon*: Executing direct acyclic graphs as parallel jobs on anything. In *2018 IEEE/ACM Workflows in Support of Large-Scale Science (WORKS)*, pages 64–73. IEEE, 2018.

[15] Raffaele Montella, Diana Di Luccio, Pasquale Troiano, Angelo Riccio, Alison Brizius, and Ian Foster. Wacomm: A parallel water quality community model for pollutant transport and dispersion operational predictions. In *2016 12th International Conference on Signal-Image Technology & Internet-Based Systems (SITIS)*, pages 717–724. IEEE, 2016.

[16] Raffaele Montella, David Kelly, Wei Xiong, Alison Brizius, Joshua Elliott, Ravi Madduri, Ketan Maheshwari, Cheryl Porter, Peter Vilter, Michael Wilde, et al. Face-it: A science gateway for food security research. *Concurrency and Computation: Practice and Experience*, 27(16):4423–4436, 2015.

[17] Nicolas Offermans, Oana Marin, Michel Schanen, Jing Gong, Paul Fischer, Philipp Schlatter, Aleks Obabko, Adam Peplinski, Maxwell Hutchinson, and Elia Merzari. On the strong scaling of the spectral element solver nek5000 on petascale systems. *Proceedings of the Exascale Applications and Software Conference 2016*, Apr 2016.

[18] Dante D Sánchez-Gallegos, Diana Di Luccio, JL Gonzalez-Compean, and Raffaele Montella. A microservice-based building block approach for scientific workflow engines: Processing large data volumes with dagonstar. In *2019 15th International Conference on Signal-Image Technology & Internet-Based Systems (SITIS)*, pages 368–375. IEEE, 2019.

[19] Dante D Sánchez-Gallegos, Diana Di Luccio, José Luis Gonzalez-Compean, and Raffaele Montella. Internet of things orchestration using dagon* workflow engine. In *2019 IEEE 5th World Forum on Internet of Things (WF-IoT)*, pages 95–100. IEEE, 2019.

[20] Dante Domizzi Sánchez-Gallegos, Diana Di Luccio, Sokol Kosta, JL Gonzalez-Compean, and Raffaele Montella. An efficient pattern-based approach for workflow supporting large-scale science: The dagonstar experience. *Future Generation Computer Systems*, 122:187–203, 2021.

[21] C. A. Schneider, W. S. Rasband, and Eliceiri K. W. NIH Image to ImageJ: 25 years of Image Analysis. *Nature Methods*, 9(7):671–675, 2012.

[22] R. Sedona, G. Cavallaro, J. Jitsev, A. Strube, M. Riedel, and J.A. Benediktsson. Remote Sensing Big Data Classification with High Performance Distributed Deep Learning. *Remote Sensing*, 11(24), 2019.

[23] Jonathan Vincent, Jing Gong, Martin Karp, Adam Peplinski, Niclas Jansson, Artur Podobas, Andreas Jocksch, Jie Yao, Fazle Hussain, Stefano Markidis, Matts Karlsson, Dirk Pleiter, Erwin Laure, and Philipp Schlatter. Strong scaling of openacc enabled nek5000 on several gpu based hpc systems, 2021.