EuroHPC-01-2019

IO-SEA

IO – Software for Exascale Architectures

Grant Agreement Number: 955811

D1.5
Final report on applications experience

Final

Version:	2.0
Author(s):	E. B. Gregory (FZJ), P. Couvée (Eviden),
Contributor(s):	D. Caviedes-Voullieme (FZJ), D. Chapon (CEA), R. Furmanek (TU Ostrava), M. Golasowski (IT4I), J. Wong (ECMWF)
Date:	March 27, 2024

Project and Deliverable Information Sheet

IO-SEA Project	**Project ref. No.:**	955811
	Project Title:	IO – Software for Exascale Architectures
	Project Web Site:	`https://www.iosea-project.eu/`
	Deliverable ID:	D1.5
	Deliverable Nature:	Report
	Deliverable Level: PU *	**Contractual Date of Delivery:** 31 / March / 2024
		Actual Date of Delivery: XX / March / 2024
	EC Project Officer:	Rene Chatwill

* – The dissemination levels are indicated as follows: **PU** - Public, **PP** - Restricted to other participants (including the Commissions 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).

Document Control Sheet

Document	**Title:** Final report on applications experience	
	ID: D1.5	
	Version: 2.0	**Status:** Final
	Available at: `https://www.iosea-project.eu/`	
	Software Tool: LaTeX	
	File(s): IO-SEA-1.5.pdf	
Authorship	**Written by:**	E. B. Gregory (FZJ), P. Couvée (Eviden),
	Contributors:	D. Caviedes-Voullieme (FZJ), D. Chapon (CEA), R. Furmanek (TU Ostrava), M. Golasowski (IT4I), J. Wong (ECMWF)
	Reviewed by:	Christophe Laferrière (Eviden) Maike Gilliot (CEA)
	Approved by:	Exec Board/WP7 Core Group

Document Status Sheet

Version	Date	Status	Comments
0.1	15 Jan 2024	Outline approved	complete
1.0	11 March 2024	All sections ready for internal proofreading	complete
1.2	15 March 2024	Draft ready for internal review	complete
1.3	21 March 2024	1st internal review complete	complete
1.4	25 March 2024	post-1st-review edits complete	complete
2.0	26 March 2024	final draft ready for EU submission	complete

Document Keywords

Keywords:	IO-SEA, HPC, Exascale, Software, Storage, Burst buffer

Contents

List of Figures

List of Listings

Executive Summary

In the last year of the IO-SEA project, the IO-SEA Workflow Manager has been installed on the DEEP System at Jülich Supercomputer Centre, giving use case applications access to three IO-SEA ephemeral storage services hosted on the IO-SEA Prototype system (cf. D2.3, D4.3, D5.4, D5.5 and D5.6 for more details).

In this environment, the IO-SEA scientific use cases have tested their workflows to understand the advantages and challenges of running with the SBB, NFS and DASI ephemeral services. This report details the results of these tests and the experiences of each use case. We conclude with a report of our recommendations and lessons learned in the process.

1. Introduction

The final year of the IO-SEA project has brought the deployment of the prototype IO-SEA Pilot System on the DEEP system at Jülich Supercomputing Centre. The IO-SEA Workflow Manager (WFM) has been installed on DEEP, giving access to three IO-SEA ephemeral storage services hosted on the datanodes of the prototype (cf. D2.3, D3.3, D4.3, D5.4, D5.5 and D5.6 for more details).

In the summer of 2023 the first of these services the Smart Burst Buffer (SBB) became available through WFM v1.0 for testing by the five IO-SEA scientific use cases. In our deliverable report D1.4 [1], we reported on our experiences in testing the performance and usability of the SBB service, as well as the adaption of workflows to running with the WFM in the IO-SEA storage environment.

Since then the WFM has been updated to version 1.6, with new features and functionality, and two new ephemeral services have been added, the NFS service and the DASI service. Both of these allow workflow data to be stored on a dataset on the datanodes and to be managed by the WP4 long term storage environment through the Hestia API.

In this report we describe our experience running the use case workflows in this updated environment. Each use case has endeavored to test as many features as is feasible for the character of their application and workflow. We attempt to explore the usability of these features and services and understand how they can fit into not only our benchmarking test workflows, but to imagine their use in production-scale workflows if we had available a mature IO-SEA storage environment.

At this point, we do employ some quantitative benchmarking to understand the relative performance of different types of services or different configurations within a workflow.

Finally, in continuing our role in the co-design process we also provide recommendations about how the services and the WFM could be improved, from the user's point of view. We recognize that we are testing a first version of the the IO-SEA solution rather than a polished, final product. We hope that the feedback from our experiences could assist in making future extensions of this solution a useful part of an exascale HPC ecosystem.

2. User interface and ephemeral services

As presented with more details in the deliverable D2.3 [2], different versions of the IO-SEA run time enviroment have been made available for testing by the use cases on the IO-SEA Pilot system:

- **Version 1.0 of the Workflow Manager**. This version introduced the support of the first Ephemeral Service (Smart Burst Buffer) on real datanodes. The tests by use cases on this version have been reported in IO-SEA Deliverable D1.4.
- **Version 1.2 of the Workflow Manager**. This version introduced the support of datasets. The datasets remain stored as POSIX files in a user directory (no connection with the WP4 long-term storage).
- **Version 1.4 of the Workflow Manager**. This version introduced the connection to the WP4 long-term storage, as datasets are now stored through the Hestia API in the hierarchical storage infrastructure, as described in Deliverable 4.3 [3].
- **Version 1.6 of the Workflow Manager**. This version introduces a first connection to WP5 with the support of a new DASI ephemeral service. More details about DASI and workflow adaptions are available in IO-SEA Deliverable 5.6 [4].

The new features in each version have been chosen to progressively implement the main IO-SEA concepts while remaining usable for integration and testing. The focus has been made on the features themselves and not necessarily on the ease of use, on the performance or on the robustness.

For each version, a How-To document has been provided with examples and real execution traces to help users update their environment, and a high priority has been put on supporting use cases in their integration of the new services with email exchanges, calls and shared debug sessions.

3. Use case testing goals and methods

With WFM version 1.6 there are three ephemeral services, SBB, NFS, and DASI, available on the IO-SEA Pilot System on DEEP through the IO-SEA environment. In this environment the use cases have attempted to use them all and assess usability and usefulness.

Interacting with user data through the IO-SEA WFM requires a different human workflow for users accustomed to accessing data directly through POSIX file systems and managing HPC jobs directly through the scheduler.

The eventual payoff for this learning curve will be increased performance and easier data management in the exascale and beyond when large I/O operations and datasets are the rule.

With the current WFM and the limitations of the DEEP systems it is not possible to run huge production-scale scientific workflows, but our short testing period has allowed us to make some extrapolations about how we might use the IO-SEA storage environment and services, and also to provide feedback about useful ways the user interfaces could evolve in the future.

In Chapter 4 — Chapter 8, we report the specific experiences of the five scientific use cases of the IO-SEA project. For the readers convenience we begin each chapter with a short overview of the use case workflow. In subsequent sections, we present the testing schemes and results for each ephemeral service with that workflow.

In the tests and analysis presented in this part of the report, we attempt to determine :

- Which ephemeral services are usable by the use case applications in the present deployment?
- Which of these services, or combination thereof might form part of a production scale workflow in a hypothetical mature IO-SEA deployment?
- Are there aspects of the services or WFM API which could be improved?

We have focused less on quantitative performance, as this can depend on a number of factors and, as has been discussed in IO-SEA D1.4 [1], the IO-SEA Prototype, and the DEEP system are both composed of predominantly older hardware and their purpose is for testing the concepts and the approach. Nevertheless, we occasionally report timing data when it helps to illustrate interesting behavior.

As with the testing described in our report IO-SEA D1.4 [1], the use case teams found JUBE a useful tool and much of the testing was carried out within JUBE benchmark frameworks developed with the assistance of our benchmarking team Task 1.2 and described in IO-SEA D1.2 [5] and further in D1.4. For each workflow, template batch scripts and WDF files are provided and JUBE substitutes in specific values of parameters and file system paths. In this way the tester can quickly toggle between workflow execution models: traditional or with one or more of the IO-SEA ephemeral services. Using JUBE also allowed for quick, repeatable workflow tests to check the stability of the IO-SEA Prototype system.

4. Astrophysics results

RAMSES is an open-source simulation code for astrophysical compressible plasma flows, featuring self-gravitation, magnetism, and radiative processes. It is based on the Adaptive Mesh Refinement (AMR) technique on a fully-threaded graded octree. It is written in Fortran 90 and makes intensive use of the MPI library.

4.1. RAMSES workflow overview

The RAMSES benchmark workflow illustrates the case where checkpointed data is written to disk and then read in and re-used later in the workflow. A simplified schematic of the data flow is depicted in Figure 1. In Step 1, a new simulation run is started and checkpoint data (HProt) is then repeatedly written to disk with the Hercule [6] [7] parallel I/O library before the simulation is stopped. In Step 2, the latest checkpoint data is read back to restart the simulation and scientific analysis backup data (HDep) are then repeatedly written to disk by Hercule. These two steps allow us to not only test the different Hercule dataflows implemented in RAMSES (HProt and HDep) but also to mix read and write operations in a typical RAMSES run.

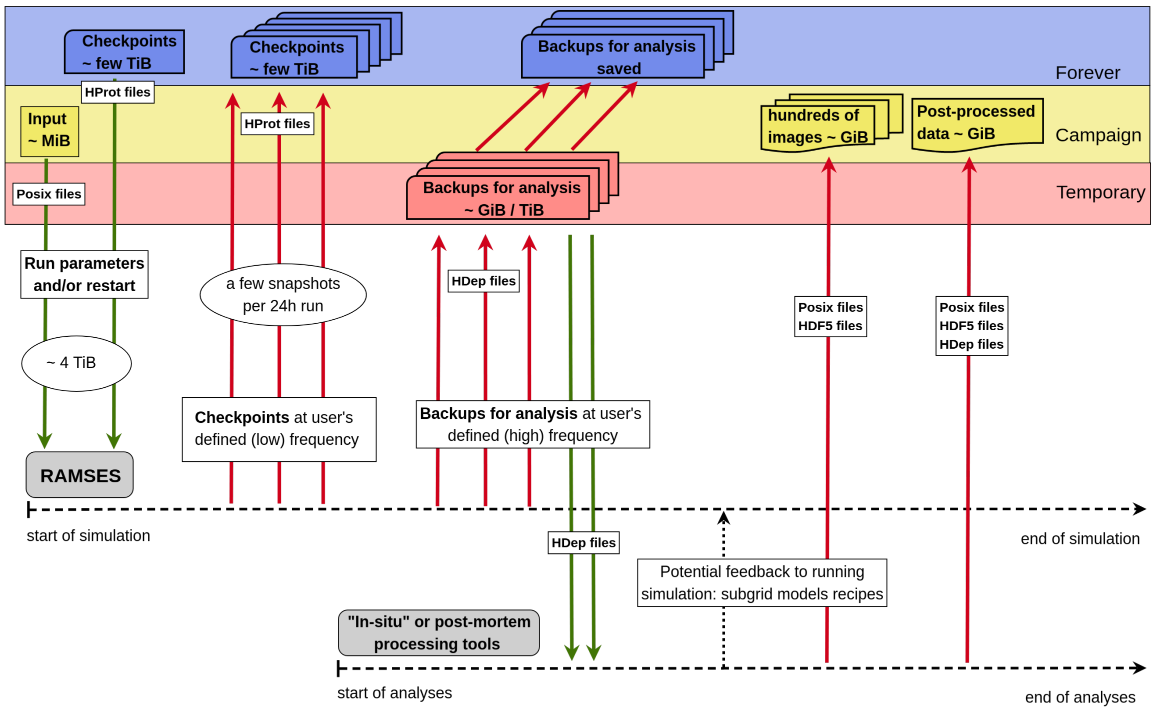

Checkpoints ~ few TiB
Checkpoints ~ few TiB
Backups for analysis saved
Forever
HProt files
Input ~ MiB
HProt files
hundreds of images ~ GiB
Post-processed data ~ GiB
Campaign
Posix files
Backups for analysis ~ GiB / TiB
Temporary
Run parameters and/or restart
a few snapshots per 24h run
HDep files
Posix files HDF5 files
Posix files HDF5 files HDep files
~ 4 TiB
Checkpoints at user's defined (low) frequency
Backups for analysis at user's defined (high) frequency
RAMSES
start of simulation
end of simulation
HDep files
Potential feedback to running simulation: subgrid models recipes
"In-situ" or post-mortem processing tools
start of analyses
end of analyses

Figure 1: The RAMSES-Hercule I/O benchmark workflow.

During the first RAMSES run, the 3D adaptive mesh and the hydrodynamical quantity scalar fields defined on that mesh are written to disk repeatedly (four times in this benchmark) to mimic the

I/O-intensive scheduling of checkpoint data occurring in a typical RAMSES production run. Two consecutive checkpoints are separated by a delay to mimic the computation between checkpoints. In this specific setup, a RAMSES checkpoint is $\sim$ 11 GB in size. In total, the first step of the workflow corresponds to a total of 44 GB of write operations. During the second RAMSES run, the most recent checkpointed data from the first step is read back from disk and the time integration of the 3D model is resumed from that point.

Assuming a scientifically interesting point is reached during that simulation, several backups ($\sim$ 6 GB each) are stored to disk for scientific analysis purposes, again repeatedly (here six times), to mimic a typical RAMSES production run. Two consecutive backups are also separated by a delay to mimic computation. During this second run, 11 GB (HProt AMR + Hydro) are read from the filesystem and $\sim$ 36 GB of backup data (six HDep outputs of $\sim$ 6 GB each) is written back to disk. This second step mixes read and write operations. In total, the workflow executes 80 GB of write operations and 11 GB of read operations.

4.2. RAMSES in the IO-SEA environment

For our benchmarking framework, we use JUBE [8]. We adapted our IO-SEA workflow manager and our JUBE benchmarking scripts from the version described in Deliverable 1.4 [1] to test the Smart Burst Buffer (SBB) ephemeral service, the NFS ephemeral service and the DASI interface.

4.2.1. SBB service

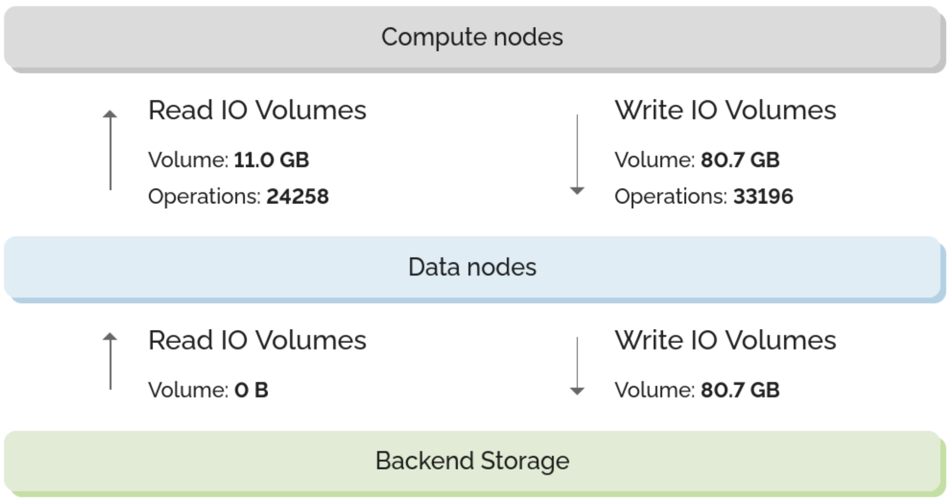


Figure 2: Overview of the size of data transferred through the data nodes between the compute nodes and back-end storage in the RAMSES-Hercule I/O benchmark with the SBB ephemeral service enabled (IO-Instrumentation web application).

Since the previous report [1], we upgraded the version of the WFM used to run the Smart Burst Buffer ephemeral service benchmark from v1.1.0 to 1.6.0. The exact same version of the workflow description file was used and we investigated further the I/O slowdown issue we encountered with the early version of the WFM prototype.

As shown in Figure 2, we still put the SBB ephemeral service to good use when trying to reuse the data stored in the datanode memory in a previous workflow step without reading data from the backend storage.

However, the use of the WFM v1.6.0 did not improve the I/O performance compared to the results presented in the previous report [1], as shown in the Figure 3. The four HProt write operations still do not appear as clearly separated I/O peaks.

We have spent some time investigating this and the root cause is not obvious. The SBB statistics collected with IO Instrumentation did not show any overload on the datanodes . We theorize that the some combination of the InfiniBand configuration or the older hardware of the datanodes could contribute to occasional unstable behavior. It is also possible that is it due to some unexpected behavior of the Hercule library.

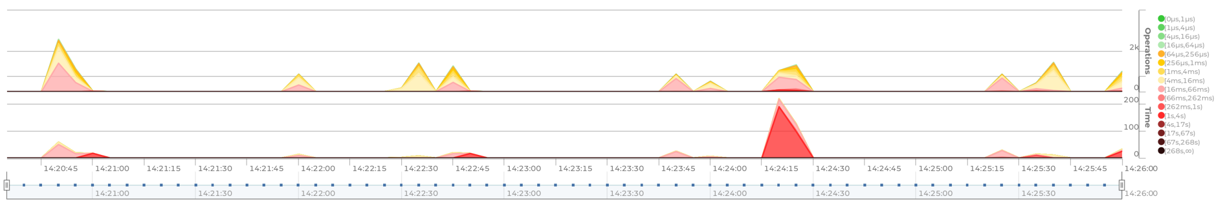


Figure 3: Timeline of the RAMSES-Hercule checkpoint write operations benchmark with the SBB ephemeral service using the WFM v 1.6.0 (IO-Instrumentation web application).

4.2.2. NFS Service

We tested the NFS ephemeral service on a RAMSES use case to store the output (HProt and HDep) of our entire workflow into a single dataset. The workflow description file template used to run the benchmark is shown in Listing 4.1.

Due to storage size limitations in the NFS ephemeral service prototype implemented on the DEEP system, we could not replicate the RAMSES run configuration tested with the SBB ephemeral service, which was using ∼80 GB of storage space. We scaled down our RAMSES AMR grid configuration by a single level to reduce the overall data volume produced during the workflow to ∼10 GB. The storage size of the NFS ephemeral service was set to 20GB to handle RAMSES I/O for this workflow.

The observed problems with large size datasets is related to utilities used to migrate the dataset through the long-term storage through the HESTIA API. This issue should disappear in WFM v2.X, as files will be stored as seperate HESTIA objects, rather than the the whole dataset as one object.

The NFS ephemeral service `mountpoint` and `namespace` variables in the workflow description files were set by the JUBE script. The value for the `namespace` attribute needs to be prefixed with `HESTIA@` to make use of the HESTIA API ([3][2]). The `namespace` path should be set to a filename somewhere in the `/afsm` filesystem. For the HESTIA API to be able to mount the dataset, the `mountpoint` directory should be created before the session startup and placed in the filesystem hierarchy with read and execution privileges for the root user.

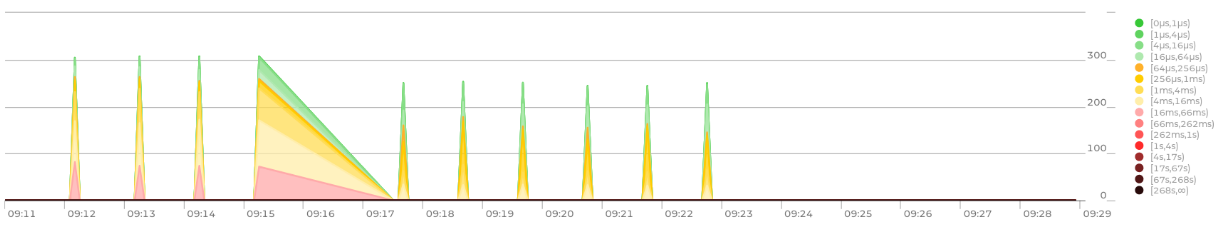


Figure 4: Timeline of the RAMSES-Hercule workflow operation benchmark with the NFS ephemeral service (IO-Instrumentation web application).

Upon completion of the workflow, the NFS `mountpoint` directory is unmounted and the dataset file (73 bytes) only contains the HESTIA dataset id, ready to be used in another session to reload the NFS filesystem.

The performance comparison with AFSM or SBB ephemeral service would not be very relevant here since the RAMSES run configuration was downscaled and the data volume reduced by a factor $\sim$8, however the write operation I/O rate appears to be very similar to the one obtained with AFSM filesystem. Figure 4 shows the timeline of the write operations for the entire workflow using the NFS ephemeral service.

At the scale of this test workflow there was no evidence of the I/O slowdown seen in the SBB RAMSES tests.

4.2.3. DASI service

A specific plugin for the DASI backend needed to be developed in Hercule to be used from our RAMSES workflow. This was done in IO-SEA WP5 and is described in IO-SEA Deliverable report D5.6 [4]. We adapted our JUBE benchmarking script to use the DASI ephemeral service in our RAMSES workflow, as shown in Listing 4.2.

The `dasiconfig` attribute of the ephemeral service configuration needs to be set to the path of the DASI configuration file displayed in Listing 4.3.

The DASI configuration file defines a `schema` attribute, set to the path of the RAMSES-specific DASI schema, shown in Listing 4.4.

This schema allows us to identify in a unique manner a subset of data written by the Hercule library. In our case, the identifier is as follows :

- `Database`: type of database, `HProt` for checkpoint/restart outputs, `HDep` for analysis purposes.
- `iTime`: output number, corresponding to different time steps in the simulation run.
- `sDom`: sub-domain in the geometric domain decomposition. In RAMSES, the subdomain number corresponds to the MPI rank of the process.
- `mesh`: RAMSES AMR mesh attributes.
- `field`: RAMSES AMR scalar fields.

```
workflow:
  name: Ramses-Workflow_NFS

services:
  - name: ramses-nfs
    type: NFS
    attributes:
      namespace: HESTIA@__NFS_NAMESPACE__
      mountpoint: __NFS_MOUNTPOINT_DIRECTORY__
      storagesize: 20GiB
      datanodes: 1
      location: dp-cn

steps:
  - name: run_init
    command: "sbatch __NFS_SUBMIT_JOB_SH_PATH__"
    services:
      - name: ramses-nfs

  - name: run_restart
    command: "sbatch __NFS_SUBMIT_RESTART_JOB_SH_PATH__"
    services:
      - name: ramses-nfs
```

Listing 4.1: WDF template, `ramses_wdf_nfs.yml.in`, for JUBE execution of the RAMSES workflow using the NFS epehmeral service.

Since the Hercule library is handling the DASI interface without any modification in the RAMSES source code, the use of the DASI ephemeral service should be completely transparent for the RAMSES user and only impact the Hercule database file name conventions.

Unfortunately, the DASI plugin prototype for Hercule was partly developed and tested on a development environment by WP5 but could not be deployed on the DEEP system before the end of this project and we are not able to provide the reader any feedback on its performance and practical use in the context of the RAMSES use case.

4.3. Conclusions

The use of the NFS, DASI, SBB ephemeral services in the context of the IO-SEA project for the RAMSES use case has been quite instructive.

Unfortunately, we have not had the time in the context of the project to do more testing of the I/O timing variations seen in the SBB with the RAMSES workflow but not seen with other use case workflows). We attempted to do a comparison test with the same workflow and the NFS service, which might highlight if the particular I/O pattern of the RAMSES workflow caused problems with common hardware, such as the InfiniBand. However, the noted problems with larger-sized Hestia datasets prevented this in the time available for the project.

```
workflow:
  name: Ramses-Hercule DASI workflow

services:
  - name: ramses-her-dasi
    type: DASI
    attributes:
      namespace: HESTIA@/afsm/iosea/USE_CASES/RAMSES/DASI_datasets/__DASI__NAMESPACE__
      dasiconfig: /p/project/iosea/USE_CASES/RAMSES/iosea-benchmark/DASI/dasi-config.yaml
      storagesize: __DASI_STORAGE_SIZE__GiB
      datanodes: __DASI_DATANODE_NUM__
      location: dp-cn


steps:
  - name: run_init
    command: "sbatch __DASI_SUBMIT_JOB_SH_PATH__"
    services:
      - name: ramses-her-dasi

  - name: run_restart
    command: "sbatch __DASI_SUBMIT_RESTART_JOB_SH_PATH__"
    services:
      - name: ramses-her-dasi
```

Listing 4.2: WDF template, `ramses_wdf_dasi.yml.in`, for JUBE execution of the RAMSES workflow using the DASI ephemeral service.

However, data reuse from the SBB datanode memory while bypassing the storage back-end is an interesting feature for the RAMSES user community that actually proved functional in the context of this project.

The NFS ephemeral service was tested in the context of this project to store both the checkpoint and the analysis Hercule databases into a single dataset for our entire RAMSES workflow. The storage size limitation in the current implementation of datasets in the IO-SEA prototype is a blocking point for any practical RAMSES application, most isolated datasets are at least 100 GB or TB scale. To be able to test it with RAMSES, a down-sized version of our RAMSES workflow needed to be configured, which made the performance comparison with other tested services difficult. However, we can report quite similar performance with AFSM in terms of I/O operation rates. This storage limitation is meant to be removed altogether in future implementations, which could benefit RAMSES users in their workflow data placement (hot to cold storage) without the need to use datamovers in their workflow description file.

The DASI ephemeral service could not be tested in a practical RAMSES use case for lack of a DASI plugin deployment in the Hercule library on the Deep system, but nevertheless the integration in Hercule makes it quite transparent for RAMSES users to use the ephemeral service without modifing the RAMSES source code. Unfortunately, we could not report performance comparison between the DASI service and other tested services in the context of this project.

```
schema: /p/project/iosea/USE_CASES/RAMSES/iosea-benchmark/DASI/dasi_schema.yaml
catalogue: toc
store: file
spaces:
       - roots:
              - path: /afsm/iosea/USE_CASES/RAMSES/dasi_root
```

Listing 4.3: DASI configuration file, `ramses_dasi_config.yml`, for execution of the RAMSES workflow using the DASI ephemeral service

```
[Database, [ iTime,[ sDom [ mesh?, field? ] ] ]
```

Listing 4.4: DASI schema, `ramses_dasi_schema.yml`, for execution of the RAMSES workflow using the DASI ephemeral service

5. CryoEM results

5.1. CryoEM workflow overview

The Figure 5 shows a simplified version of the Cryogenic Electron Microscopy (CryoEM) benchmarking workflow, including integration with JUBE benchmarking framework [8], and DASI as the data access interface and CryoEM tools. The benchmark is implemented as JUBE script which then launches the CryoEM pipeline implemented in Python.

The pipeline implements several consecutive steps, however from the workflow point of view, all these steps are being executed inside the Python pipeline in a single batch job described in the WDF file.

In the first step the pipeline pulls input data from the DASI and starts executing several CUDA binaries which process the individual TIFF images (approx. 150 MB each) produced by the microscope. Each step produces a plaintext file, its content is then used to identify the scanned sample in the images. These steps account for the bulk of the I/O load of the benchmark. The post-processing step then creates unified summary of the output log.

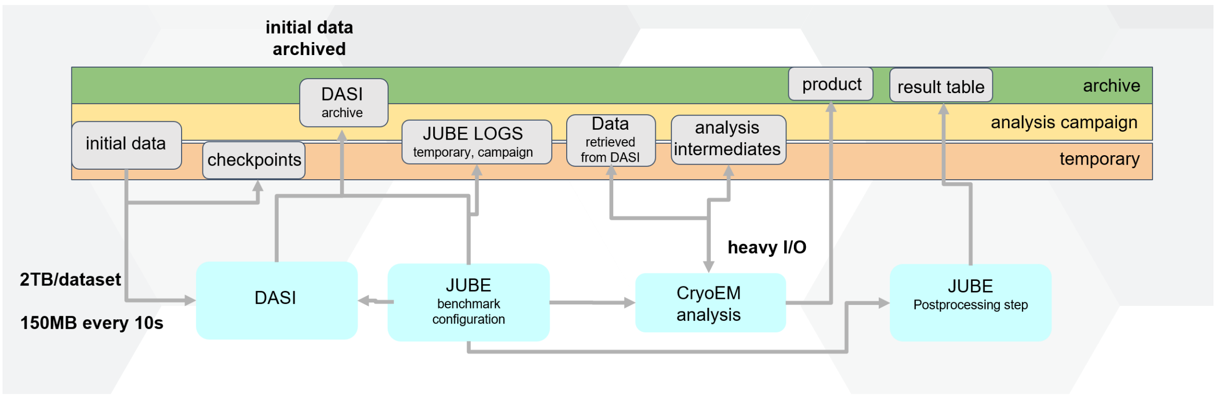


Figure 5: CryoEM benchmark workflow diagram.

5.2. CryoEM in the IO-SEA environment

5.2.1. SBB service

For the CryoEM use case, running the pipeline with SBB enabled provided inconclusive results, as is mentioned in Deliverable 1.4 [1]. During the integration of the use case, it has been discovered that sometimes the CUDA binaries had problems with writing the temporary and output files. Suspected root cause is incompatibility of the SBB service with `Mmap()` system call [9].

5.2.2. NFS Service

The CryoEM benchmarking workflow was tested using the NFS ephemeral service. The WDF was seamlessly adapted to function with the NFS service, based on configuration of SBB WDF file. In Listing 5.1, a sample CryoEM WDF specifically tailored for the NFS service is provided.

In runs with the NFS service and dataset, we did not experience the problems with the `mmap()` calls that prevented successful use of the SBB service.

```
workflow:
   name: Workflow_CRYOEM

services:
   - name: cryoem-nfs1
     type: NFS
     attributes:
        namespace: #NAMESPACE#
        mountpoint: #MOUNTPOINT#
        storagesize: #STORAGE_SIZE#
        datanodes: 1
        location: #SLURM_QUEUE#
steps:
   - name: step_1
     command: "#CRYOEM_SUBMITCMD#"
     services:
        - name: cryoem-nfs1
```

Listing 5.1: WDF `cryoem_wdf_nfs.yml` for execution of CryoEM workflow steps with the NFS service.

The attributes marked with "#" symbols are substituted by JUBE during the benchmark execution, incorporating paths specific to each run. The `namespace` attribute value is prepended with `HESTIA@` to leverage the functionalities of the HESTIA API [3][2]. The `storagesize` attribute is different according to the size of the input dataset.

JUBE [8] was integrated to automate and standardize the execution of our CryoEM benchmark. This encompassed both scenarios: with and without WFM services, as detailed in D1.4[1]. Through the utilization of JUBE, repeated iterations of the workflow were conducted, enabling a comparative analysis of the impact of different services on performance.

Within the NFS, there are three ways to get data into dataset:

- **Interactive Environment:** The command `iosea-wf access` can be used for access to the interactive environment, followed by the utilization of the `cp` command, for example.
- **Session-Based Command:** Alternatively, within a session, a command can be run with the prefix `srun -N 1 -n 1`. For instance, this approach is suitable for executing the e.g. `cp` command.
- **Data Migration:** Another option is to use the `iosea-migrate` command, which allows to populate data before starting session.

We have explored all these approaches. For this specific use case, it would be appropriate to utilize `iosea-migrate` for a one-time data computation. Alternatively, you can consider using the DASI, which is described in the following section.

5.2.3. DASI service

The adaptation of the CryoEM workflow to use DASI for retrieving and archiving data is described in detail in IO-SEA Deliverables D5.3 [10] and D5.4 [11]. The adaptation required changes in the WDF YAML file, which is presented in Listing 5.2. It contains the new path to the dasi-config file presented in Listing 5.3. File `dasi-config.yml` refers to the schema which is described in D5.6 [4]. We have tried to compare data retrieval using the CLI DASI commands with using the DASI Python interface, which is thoroughly described in D5.3 [10]. Both approaches create structure within the DASI dataset according to DASI. Data is later retrieved from the dataset with the execution of a data retrieval command. In the case of DASI CLI, this involves using `dasi-get`, or alternatively, `retrieve_data.py` for the Python interface.

```yaml
workflow:
   name: Workflow_CRYOEM

services:
   - name: cryoem-dasi1
     type: DASI
     attributes:
        namespace: #NAMESPACE#
        dasiconfig: #DASI_CONFIG#
        storagesize: #STORAGE_SIZE#
        datanodes: 1
        location: #SLURM_QUEUE#
steps:
   - name: step_1
     command: "#CRYOEM_SUBMITCMD#"
     services:
        - name: cryoem-dasi1
```

Listing 5.2: WDF `cryoem_wdf_dasi.yml` for execution of CryoEM workflow steps with the DASI NFS service.

In our use case, we use DASI as a store to return data. Data are processed and gradually moved during processing. When using NFS, the data from the POSIX system is moved using native DASI interface in Python, directly to the dataset on NFS. In this case, we first create a DASI store directly in the dataset, as when using the dasi-put command, which creates a DASI store directly in the NFS service dataset.

Within step_1 are integrated steps for `dasi-put` and `dasi-get`, described below,

```
srun -N 1 -n 1 dasi-put --config=dasi-config.yml ${DASI_DATA} /input/folder/data_50s.tar
srun -N 1 -n 1 dasi-get --config=dasi-config.yml ${DASI_DATA} /output/data_50s.tar
```

or commands for run `archive_data.py` and `retrieve_data.py`.

```
srun -N 1 -n 1 /archive_data.py -p 50s -c /dasi-config.yml
srun -N 1 -n 1 /retrieve_data.py -p 50s -c /dasi-config.yml
```

Before initializing wfm-api, it is necessary to activate source `/p/scratch/iosea/DASI/activate.sh` to make DASI commands available. These commands use the path to the `dasi-config.yaml` file, and the set of keywords `${DASI_DATA}` that are part of the schema. This creates a key to return the relevant data.

In the case of using the integrated DASI commands, we were able to upload our files only as one package of files, using a tarball. However, this approach incurs overhead in having to use the untar operation on archived data, as can be seen in Table 1.

When using a native Python DASI interface, compared to the NFS service, it saved an average of 1.4 seconds, and compared to returning data as a tarball, it is more than 10 seconds. Slight acceleration with DASI Python interface than with NFS, could be explained by the fact that the data are stored for DASI directly within the dataset, while with NFS it is moved from the POSIX system to the NFS dataset, where it is subsequently processed. However the approach with native DASI interface does not keep the original modification and access times of the input files stored in DASI dataset. The CryoEM pipeline in its current implementation relies on them, as the individual images are generated sequentially by the microscope and the processing pipeline uses the time information for scheduling purposes. We have modified the times manually so that the original ordering of the images is kept.

```
---
  schema: #DASI_SCHEMA#
  catalogue: toc
  store: file
  spaces:
   - roots:
      - path: #DASI_ROOTS_PATH#
```

Listing 5.3: WDF `dasi-congig.yml.in` CryoEm DASI configuration file.

5.3. Conclusions

We have successfully integrated the CryoEM use case in the IO-SEA WFM up to version v1.6 and verified its operation in various modes on the DEEP machine.

The main takeaway from the CryoEM use case is that it can benefit from the DASI integration, mainly for the metadata features used to describe the dataset, however the schema can be improved to better describe the flat dataset with many files inside. Main obstacle was the DASI ephemeral service, which is designed to work with single files as objects, which proved impractical for the use case, which relies on a dataset consisting up to a thousand files. Therefore we used a native Python interface of the DASI library, which managed the files as individual objects.

		None avg [std] (s)	**NFS** avg [std] (s)	**DASI python** avg [std] (s)	**DASI tar** avg [std] (s)
Preprocess	**DASI retrieve**	7.2 [1.4]	17 [0.8]	15.6 [0.2]	26 [1.7]
Processing of 1 Image	**Transfer data**	0.3 [2.2]	0.1 [0.1]	0.1 [0.1]	0.1 [0.1]
	SW MotionCore	4.5 [0.4]	4.3 [0.2]	4.3 [0.1]	4.4 [0.1]
	SW GCTF	2 [0.4]	1.8 [0.1]	1.8 [0.1]	2 [0.1]
	Process one image	7.4 [2.5]	6.7 [0.2]	6.6 [0.1]	7.1 [0.2]
Processing of entire dataset	**Whole pipeline**	888.3 [6.7]	871.4 [0.4]	871.8 [0.5]	872.6 [0.4]

Table 1.: Statistics for different services, with individual parts of pipeline for CryoEM benchmark.

The SBB integration proven infeasible due to incompatibility of the CUDA binaries with the SBB. NFS integration brings slight improvement of the overall run-time of the pipeline as visible in Table 1. The code of this application is not a typical HPC application, but rather a streaming pipeline designed for continuous processing, however even with this difference it can benefit from the IO-SEA HSM as well as from the workflow management features.

Based on experience, it is advisable to use the `iosea-wf status` command before starting a new session. This helps to clean the database. Occasionally, the dataset may still be allocated from the previous session even though it has already been destroyed.

6. Weather forecasting results

In this section we summarise the results and experiences of running ECMWF's weather forecasting workflow with the ephemeral services available in the IO-SEA workflow manager environment and any future improvements that would be useful for our workflow.

6.1. ECMWF workflow overview

The weather forecasting benchmark developed in the IO-SEA project has been created to mimic the I/O patterns of ECMWF's operational weather forecasting model. The benchmark simulates the writing of ensemble forecast outputs to ECMWF's FDB object store [12] and the downstream post-processing of these outputs in product generation. The workflow, depicted in Figure 6, involves running multiple I/O simulator instances in parallel to replicate the writing of model outputs at each step by different ensemble members and the triggering of product generation at specific steps when the data required from all members is available. The simultaneous reading and writing of data in ECMWF's weather forecasting workflow results in high contention for I/O resources, a bottleneck in the operational workflow whose time-critical component is constrained to complete within one hour.

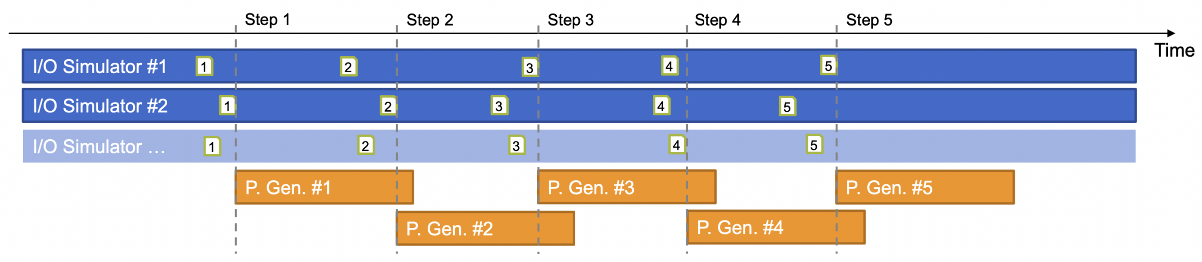


Figure 6: Timeline of the ECMWF benchmarking workflow. Each box is a job submitted to the queueing system. When all the I/O simulator instances have issued the completion notification for a particular step, the submission of the corresponding product generation job is triggered.

In this report, we describe results from running a workflow with five I/O simulator instances followed by seven product generation jobs, the same configuration discussed in IO-SEA Deliverable 1.4 [1]. We use the version of the workflow that has been adapted to use DASI for the writing and reading of model outputs, which has been described in IO-SEA Deliverables 5.3 [10] and 5.4 [11].

6.2. ECMWF in the IO-SEA environment

The integration of the WFM into the benchmark, described in [1], involved the definition of a new execution context in Kronos [13], a workflow scheduler used to submit the jobs in the benchmark taking into account job dependencies. The new context submits jobs using the `iosea-wf run` command instead of `sbatch` in addition to starting and stopping the workflow sessions. In this section we summarise the results of experiments with the adapted workflow and the different ephemeral services.

6.2.1. SBB Service

With the SBB ephemeral service, the weather forecasting workflow showed improved total read time as monitored by IO-Instrumentation but slower total write time compared with no ephemeral services. The slower write time could be a result of the old data nodes available on the DEEP system as well as the fast `/afsm` filesystem. The benefits of using the SBB service became more apparent on a loaded filesystem, which was mimicked by running a number of IOR jobs in parallel with the workflow. Improvements in both the total read and write time were observed compared to runs without the WFM when four IOR jobs where running in parallel. Further increasing the number of IOR jobs up to seven resulted in greater improvements in the total read and write time observed from using the SBB ephemeral service, and a more detailed discussion of the weather forecasting workflow with the SBB service can be found in [1].

6.2.2. NFS Service

```
workflow:
   name: ecmwf-nfs-workflow

services:
   - name: ecmwf-nfs
     type: NFS
     attributes:
        namespace: @DATASET@
        mountpoint: @FDB_ROOT@
        storagesize: 50GiB
        datanodes: 1
        location: dp-cn

steps:
   - name: multiohammer
     command: "sbatch {{ job_script_path }}"
     services:
        - name: ecmwf-nfs
   - name: pgen
     command: "sbatch {{ job_script_path }}"
     services:
        - name: ecmwf-nfs
```

Listing 6.1: WDF `ecmwf_wdf.yaml.` for execution of ECMWF weather forecasting workflow steps with the NFS service.

For the NFS ephemeral service, the workflow definition file was adapted to take the form shown in Listing 6.1. The attributes enclosed in "`@`" symbols are replaced by JUBE during the benchmark run with the paths unique to the run. The value for the `namespace` attribute is prefixed with `HESTIA@` to make use of the HESTIA API [2, 3]. As all the data written and read in during the weather forecasting workflow is produced during the workflow run, it was not necessary to access the dataset interactively using the `iosea-wf access` command to populate the dataset with initial data or modify it between steps.

In ECMWF's operational setting, the model and post-processing outputs are stored in the FDB and also archived to tape, where the data is available for around a day in the FDB before being wiped. During this time the data is available for additional processing pipelines and users to access and it might perhaps be a bit cumbersome in the form of a dataset, which can only be accessed through a workflow session. In particular, if users are only interested in querying the data available. In terms of how data would be split into datasets, ECMWF runs four operational forecasts each day at 00, 06, 12 and 18. We would expect the data from each production run to form a dataset and as such the datasets would be identified by the datetime stamp of the run.

6.2.3. DASI Service

The adaption of the weather forecasting workflow to use DASI for retrieving and archiving data is described in detail in IO-SEA Deliverables D5.3 [10] and D5.4 [11]. As the original workflow used ECMWF's semantic data store, the FDB, on which DASI is based, the adaption to use DASI did not require extensive application changes. The schema and configuration files for DASI, shown in Listings 6.2 and 6.3, respectively, re-use the existing files for the FDB.

```
catalogue: toc
store: file
schema: "@CONFIG_DIR@/schema"
spaces:
- roots:
  - path: "@FDB_ROOT@"
```

Listing 6.2: Weather forecasting workflow's DASI configuration file

For the WDF, we modified the file for the NFS service to include the `dasiconfig` attribute and removed the `mountpoint` attribute, which is now retrieved from the roots specified in the DASI configuration file. In the specification of the `namespace` attribute, which should now point to a directory, we prefix with `HESTIA@` similarly to the NFS service runs to make use of the HESTIA API.

6.3. Conclusions

Figure 7 shows a comparison of the IO-Instrumentation read and write times for benchmark runs with the three ephemeral services and a vanilla run with no ephemeral service. The lowest read time is achieved with the Smart Burst Buffer service, with the NFS and DASI services having slightly higher read times compared to the vanilla benchmark run. On the other hand, the write times are similar for

```
[ class, expver, stream=enfo/efov, date, time, domain

      [ type=tu, levtype, reference
             [ step, number, levelist?, param ]]

      [ type, levtype
             [ step, quantile?, number?, levelist?, param ]]

]

[ class, expver, stream=waef/weov, date, time, domain
      [ type, levtype
             [ step, number?, param, frequency?, direction? ]]
]
```

Listing 6.3: Section of the weather forecasting workflow's DASI schema

the NFS, DASI and no ephemeral service runs and the SBB service performance is the longest. This can be seen more clearly in Table 2, which contains the data presented in Figure 7. The possible reasons behind the poor performance of the SBB service when writing was discussed in Section 6.2.1. In particular, the benefits of the SBB service is evident when the filesystem is under heavy load, which is not shown in Figure 7.

	None	SBB	NFS	DASI
Write Time (s)	22.88	150.05	24.84	22.49
Read Time (s)	30.50	17.23	52.73	45.68

Table 2.: Comparison of IO-Instrumentation total read and write times in the ECMWF workflow for different ephemeral services.

Overall, the ECMWF weather forecasting workflow would benefit from the SBB ephemeral service due to the heavy data re-use in the post-processing part of the workflow, which reads in 70% of the data written by the I/O simulators. In addition, the use of the DASI ephemeral service would allow the workflow to make use of the backend management provided. One of the advantages of the WFM, is the relatively straightforward integration of the WFM into the benchmark while enabling the use of many different services.

It would have been interesting to run experiments using multiple services simultaneously and potentially with data movement between different storage tiers. In the weather forecasting workflow, the post-processing is triggered on the completion of particular steps of the model and after its completion the data for those steps could potentially be moved to colder storage tiers.

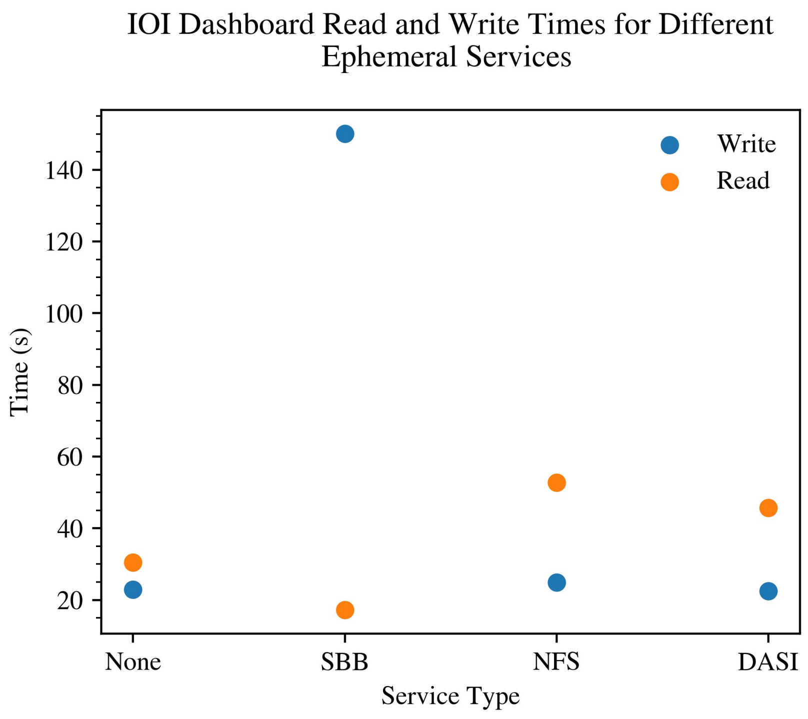


Figure 7: Comparison of IO-Instrumentation total read and write times in the ECMWF workflow for different ephemeral services.

7. TSMP results

7.1. TSMP workflow overview

The Terrestrial Systems Modelling Platform (TSMP) is a fully coupled, scale consistent, highly modular and massively parallel regional Earth System Model. Figure 8 shows a simplified workflow for TSMP, including the core sections, i.e., the simulation pipeline of the three concurrently executing component models (COSMO, ParFlow, CLM), as well as a post-processing step. The simulation pipeline has a few read operations and many write operations for model output. This output is no longer used in the simulation itself but can be ingested by the post-processing step, with potentially I/O intensive read/write operations. Each of the component models writes output independently of each other, both in terms of their generation (i.e., different binaries are generating the output) and storage (i.e., different file sets). The component models of TSMP generate output files in different formats, including ParFlow binary (PFB) and Network Common Data Format (NetCDF), at hourly intervals throughout the duration of the simulation. For testing, we have implemented an NCL script to operate on the NetCDF files, aggregating the hourly model output files into one single file for additional post-processing and archiving.

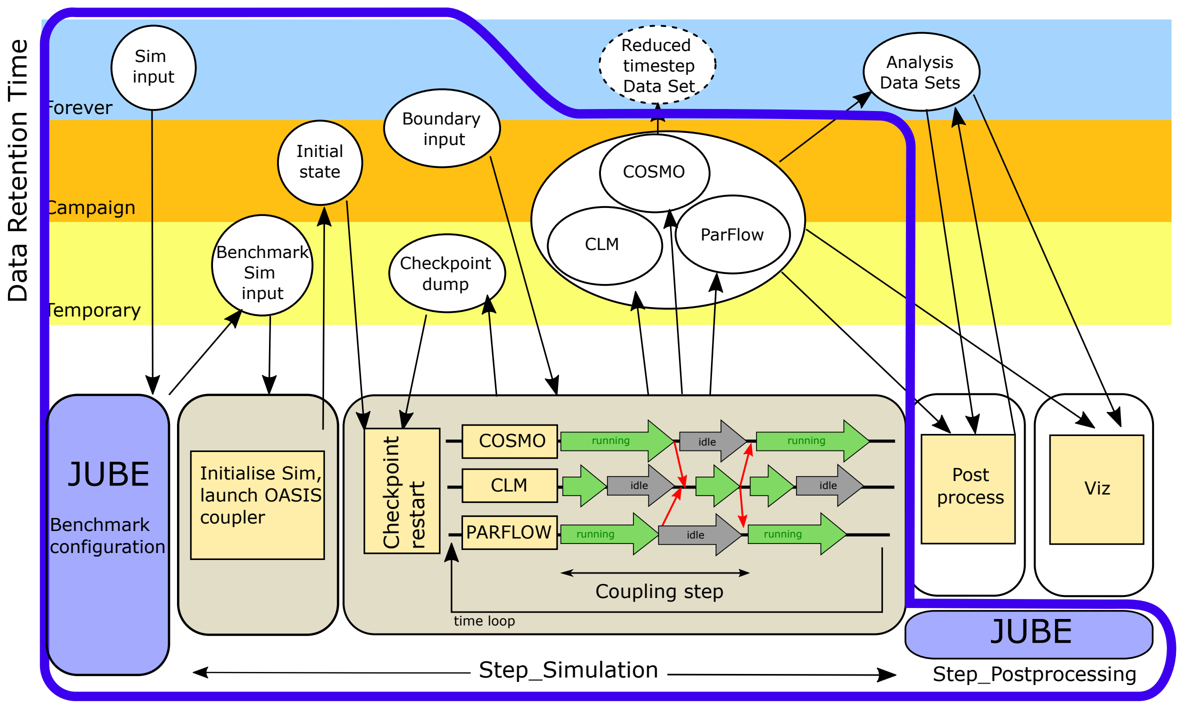


Figure 8: TSMP workflow diagram.

The workflow has been implemented within a multipurpose and flexible JUBE script, which also allows us to use DEEP-SEA developments with TSMP, notably Modular Supercomputer Architecture (MSA) job configurations. In such case ParFlow is executed on the GPU, while COSMO and CLM are executed on the CPU.

7.2. TSMP in the IO-SEA environment

The TSMP workflow described has been used to test two (SBB and NFS) of the three ephemeral services deployed in the IO-SEA test bed in the DEEP system. Currently work is being done to also test DASI. The JUBE script was extended to incorporate the IO-SEA Workflow Manager (WFM), by calling the respective yaml-format Workflow Description Files (WDF). In the sections below we describe our experience with these services.

7.2.1. SBB service and MSA

Here we summarise key points of the experience of using TSMP with the SBB ephemeral service, which was thoroughly documented in IO-SEA Deliverable 1.4 [1]. The WFD for TSMP-SBB is shown in Listing 7.1

```
---
  workflow:
    name: tsmp_idealscal_workflow
  services:
    - name: TSMP_SBB
      type: SBB
      attributes:
        targets: "/path/to/rundir/caseName"
        flavor: high
        datanodes: 2
        location: dp-cn
  steps:
    - name: step_simulation
      command: sbatch /path/to/rundir/caseName/tsmp_slm_run.bsh"
      services:
        - name: TSMP_SBB
    - name: step_postprocess
      command: sbatch /path/to/script/postp.bsh
      services:
        - name: TSMP_SBB
```

Listing 7.1: WDF for the TSMP workflow using the SBB service.

Notably, we have demonstrated the simultaneous use of the SBB ephemeral service with MSA jobs in the DEEP system, with ParFlow running on either of the GPU-accelerated modules (`dp-esb` and `dp-dam`), fully integrated into the TSMP JUBE workflow. This is a highlight, as is demonstrated a high level of integration between DEEP-SEA and IO-SEA, given that TSMP is the only application which participates in both projects.

We tested an idealised configuration for TSMP simulating 24 hours, and writing output at a frequency of one hour in simulated time. The job is run on five CPU nodes (`dp-cn` partition) — four nodes for COSMO and one for CLM — as well as one GPU node (`dp-esb` partition) for ParFlow.

Figure 9 shows the LLView usage overview for the three sub-jobs for each of the components. The LLView sampling frequency is comparatively low, and the GPU kernels are very short, which explains the low usage. The efficiency of this MSA job is not the point of interest in this exercise, and consequently it is not explored further here.

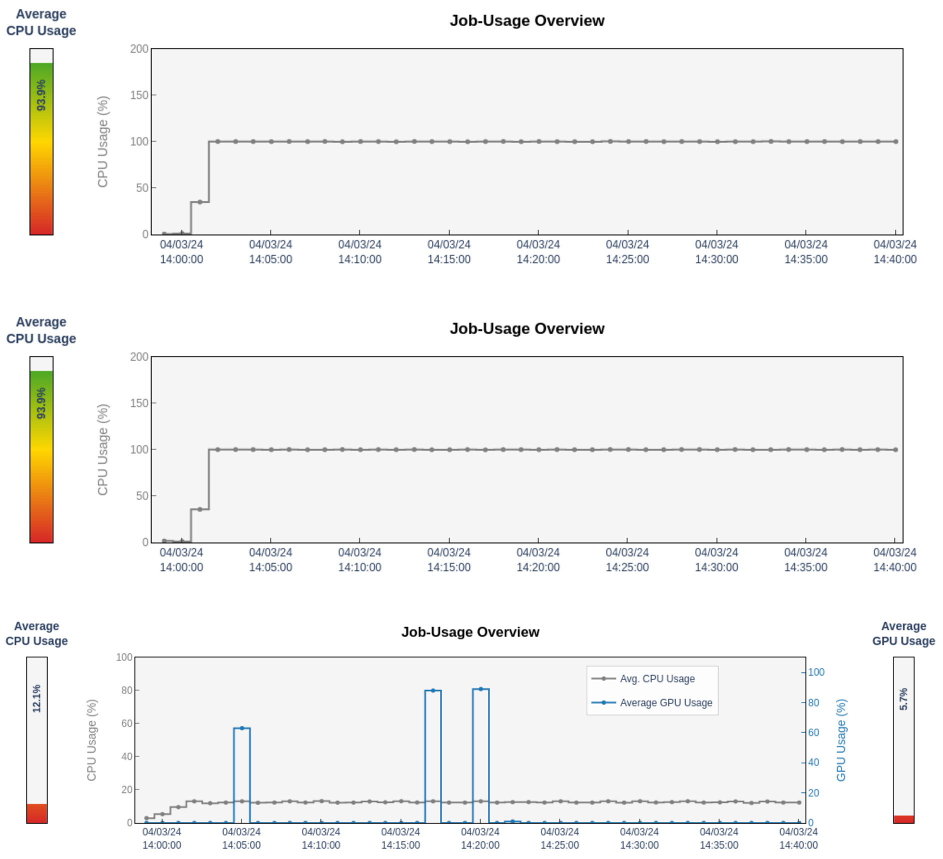
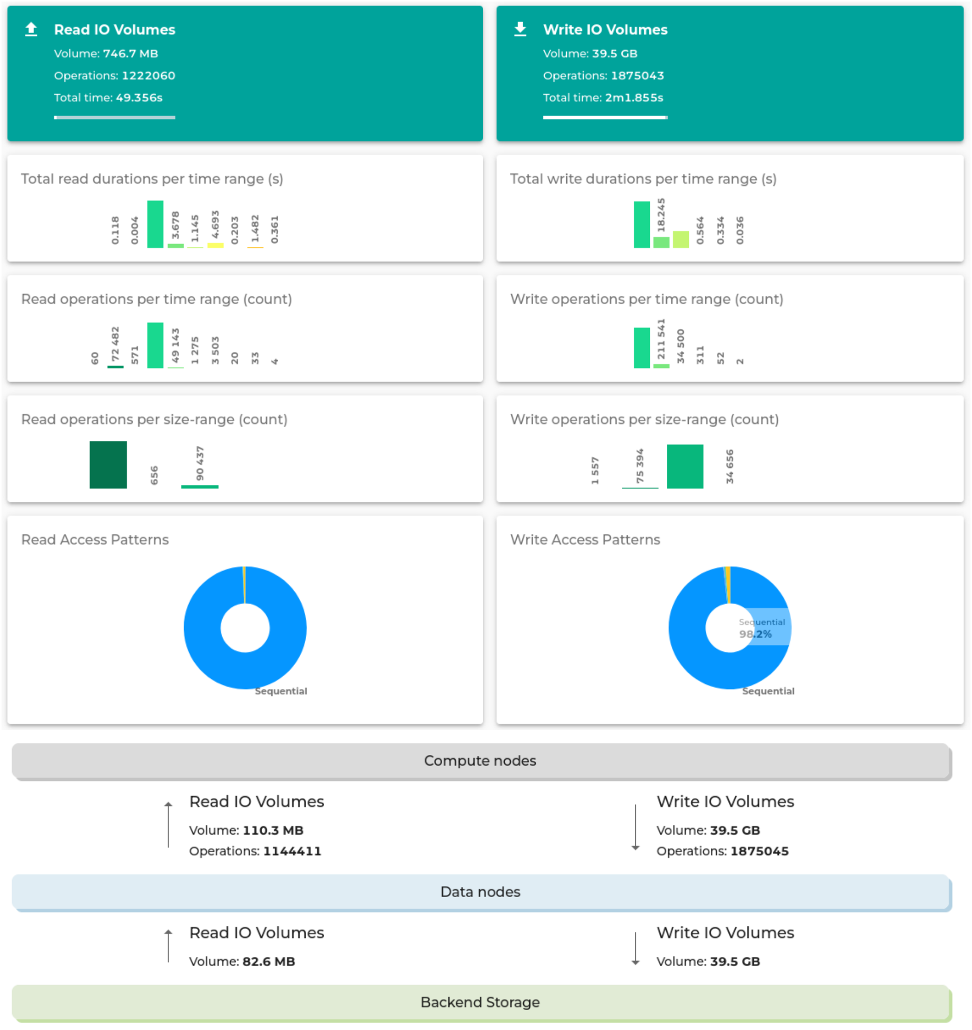


Figure 9: LLView overview of a TSMP MSA job using the SBB service, showing COSMO (top), CLM (middle) and ParFlow (bottom) processes.

Figure 10 shows the IOI overview of the TSMP MSA job. This job ran only the simulation step, which is why the read volumes are small (i.e., only reading input files for the simulation), but the write volumes are large. The post processing step should read simulation output, however this has no relation to the MSA nature of the job either, and therefore has not been the focus of this demonstration.

Figure 11 shows a selection of the IOI visualisations. Firstly, we shown the I/O operations, where it is easy to identify the 24 output events. Similarly, the distribution of the durations of write events is also shown, with most of them in the ranges between 64 and 256 μs. The key information in Figure 11 are the SBB events illustrated with the SBB clients and SBB servers. In this figure the activity of the SBB in the write events is evident, as is the fact that read events are rare in the workflow.



Read IO Volumes

Volume: **746.7 MB**

Operations: **1222060**

Total time: **49.356s**

Write IO Volumes

Volume: **39.5 GB**

Operations: **1875043**

Total time: **2m1.855s**

Compute nodes

Read IO Volumes
Volume: **110.3 MB**
Operations: **1144411**

Write IO Volumes
Volume: **39.5 GB**
Operations: **1875045**

Data nodes

Read IO Volumes
Volume: **82.6 MB**

Write IO Volumes
Volume: **39.5 GB**

Backend Storage

Figure 10: IOI overview of a TSMP MSA job using the SBB service, showing total read and write volumes.

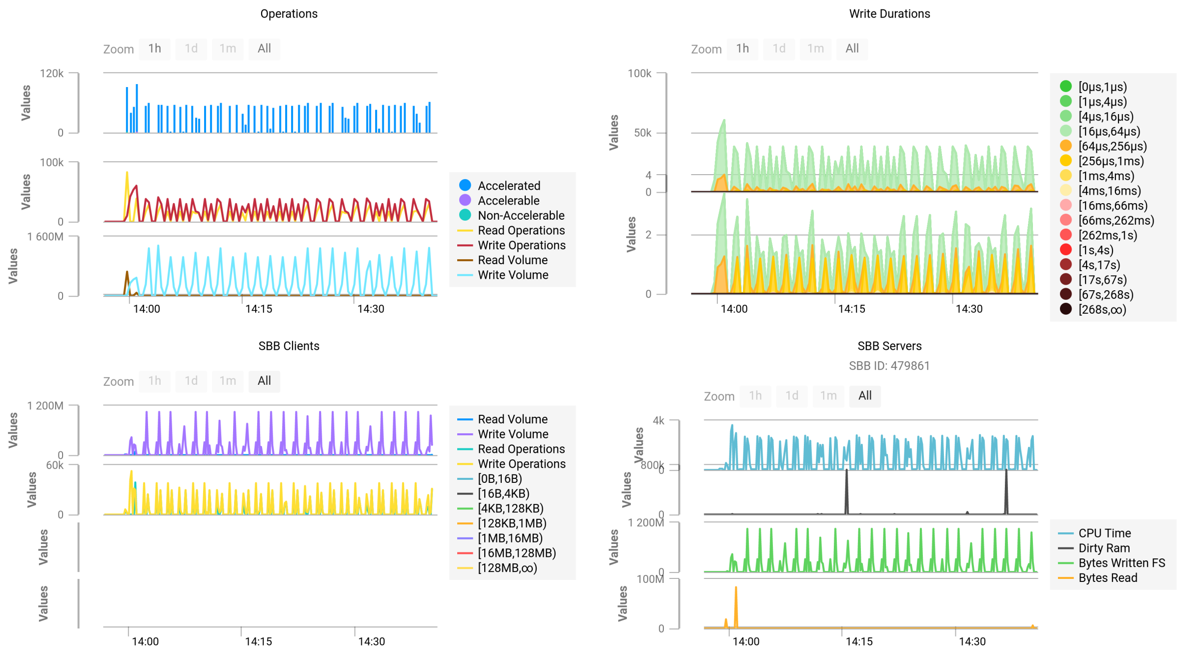


Figure 11: Selected IOI timeline visualisations for the TSMP MSA job targeting the SBB service, showing I/O operations, write durations, SBB operations and statistics of the SBB server.

7.2.2. NFS Service

Similarly to our test of the SBB service, we have been able to run MSA jobs targeting the NFS ephemeral service. The workflow is identical to the SBB workflow (and has been implemented also within the JUBE framework) with the exception that the WDF is modified to request the NFS service. A sample WDF for NFS is shown in Listing 7.2.

```yaml
---
  workflow:
    name: tsmp_idealscal_workflow
  services:
    - name: TSMP_NFS
      type: NFS
      attributes:
        namespace: "/afsm/iosea/USE_CASES/TSMPdatasets/myDataset"
        mountpoint: "/path/to/the/tsmp/setup/dir"
        storagesize: 60GiB
        datanodes: 1
        location: dp-cn
  steps:
    - name: step_simulation
      command: "sbatch /path/to/the/tsmp/setup/dir/tsmp_slm_run.bsh"
      services:
        - name: TSMP_NFS
    - name: step_postprocess
      command: "sbatch /path/to/the/tsmp/setup/dir/postp.bsh"
      services:
        - name: TSMP_NFS
```

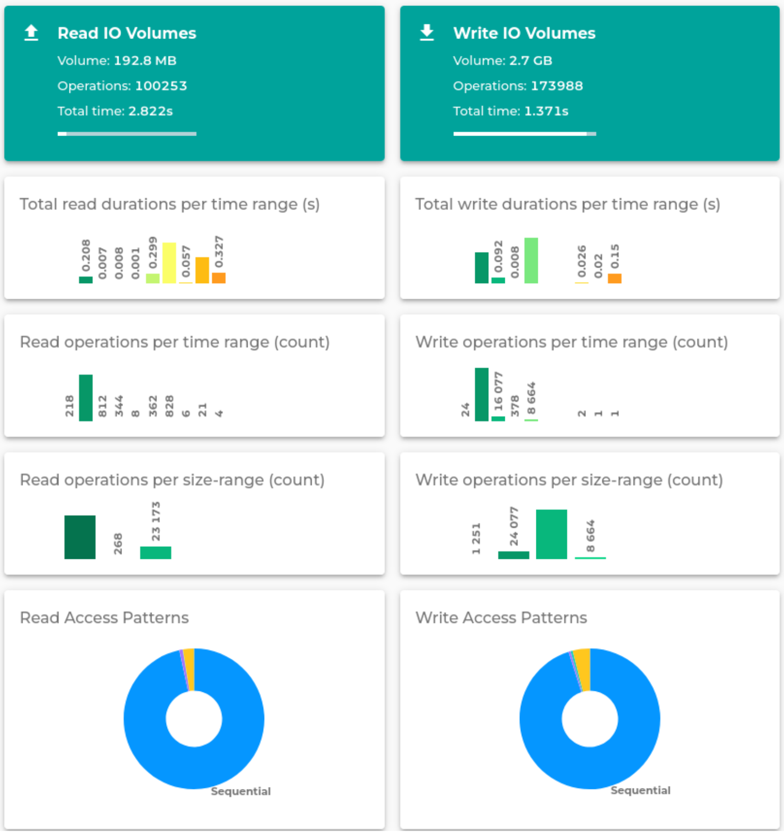
Listing 7.2: WDF for the TSMP workflow targeting NFS.

Figure 12 shows the IOI overview for a TSMP MSA job targeting the NFS ephemeral service. We tested a small idealised configuration for TSMP simulating six hours, and writing output at a frequency of one hour in simulation time. The job is run on two CPU nodes (`dp-cn` partition) — one node for COSMO and one for CLM — and also one GPU node (`dp-esb` partition) for ParFlow.

Figure 13 shows additional indicators of the I/O activity in the MSA job targeting NFS. In the operation timeline it is possible to see seven groups of writing events, which correspond to an initial checkpoint and the successive hourly snapshots for the six hour simulation. In this particular job it appears possible to distinguish writing events from the different components, which are close together but do not occur simultaneously (as there are CLM operations in between). It can be seen that the initial checkpoint is a larger event, which is reflected in the size and the duration of the writing event.

7.2.3. DASI service

TSMP set out to use the DASI service to archive data postmortem. Due to the nature of TSMP, comprised of three different solvers, each with its own I/O strategy, all aiming POSIX systems, it was not planned to make use of the DASI API natively. This requires some postprocessing to explode


Read IO Volumes
Volume: 192.8 MB
Operations: 100253
Total time: 2.822s

Write IO Volumes
Volume: 2.7 GB
Operations: 173988
Total time: 1.371s

Total read durations per time range (s)
0.208
0.007
0.008
0.001
0.299
0.057
0.327
Total write durations per time range (s)
0.092
0.008
0.026
0.02
0.15
Read operations per time range (count)
218
812
344
8
362
828
6
21
4
Write operations per time range (count)
24
16 077
378
8 664
2
1
1
Read operations per size-range (count)
268
23 173
Write operations per size-range (count)
1 251
24 077
8 664
Read Access Patterns
Sequential
Write Access Patterns
Sequential

Figure 12: IOI overview of a TSMP MSA job targeting the NFS service, showing total read and write volumes.

single snapshot files containing all variables (from each component model) so that there will be a single file per variable, per snapshot. These files can then be ingested using a suitable schema by DASI. At the time of writing of this reports, this is still under testing.

7.3. Conclusions

The key highlight is the integration of MSA jobs and the IO-SEA WFM. This is a clear cross-project collaboration between DEEP-SEA and IO-SEA, and a proof of concept of the integration of the different technologies. This is particularly relevant, because MSA jobs are expected to be essential for production jobs targeting exascale machines. Moreover, since the use of GPUs and MSA also favours simulating higher resolution problems, the use of ephemeral data services is likely to become more relevant, as the data throughput will increase and managing the output data will become more intensive.

The results from the SBB ephemeral service show that I/O times are reduced. This in turn suggests that the expected increased output volume and rate from the main simulation pipeline in next-generation TSMP simulations (e.g., over Europe at 3km resolution) may still be handled efficiently with an SBB back-end. In our benchmarking we have not pursued a full workflow analysis with SBB.

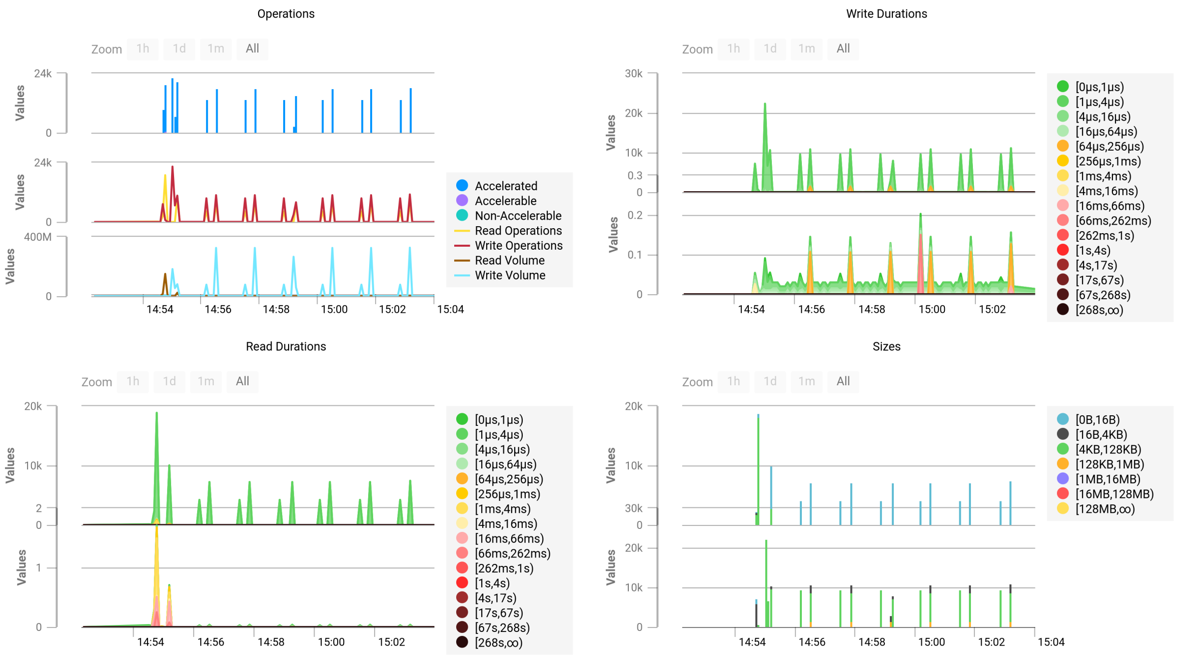


Figure 13: Selected IOI timeline visualisations for the TSMP MSA job targeting the NFS service, showing I/O operations, write and read durations, and size of the operations

However, in a production setting there is a larger amount of pre/post processing data movements that would likely also benefit from SBB, even under current production settings. SBB seems particularly useful for MSA jobs, as arithmetically intensive kernels experience significant speed-up, potentially increasing the relevance of time spent on I/O operations. A systematic comparison of SBB and NFS jobs is of interest. For this, a suitable problem size and a suitable resource configuration, in which the I/O operations represent a clear bottleneck, needs to be pinpointed. The idealised cases shown here have an I/O frequency equivalent to those of current production jobs, and not necessarily posing an I/O bottleneck.

Using the DASI service would be natural for TSMP, insofar community standards already favour data to be stored in the fashion in which DASI would ingest it. In practice, large climatology studies imply many experiments each with many ensemble members, which may have many downstream users with interest in accessing different variables, times, experiments, etc. There is an inherent difficulty in storing large datasets in such a flexible and accessible way, while maintaining an efficient use of the storage systems. That is, a large part of these datasets should be archived, but at the same time, large parts need to remain easily accessible. We see DASI as a very clear way forward to enable a low-overhead management of the datasets, which maintain transparent scientific language (semantically) and provides access to lower level storage solutions. Moreover integration with the other ephemeral services would likely provide added value, reducing I/O times, allowing to post-process the raw output on the fly and ingest it by DASI.

8. LQCD results

8.1. LQCD workflow overview

Lattice quantum-chromodynamics (LQCD) is a non-perturbative, numerical Monte-Carlo approach to calculating physical properties of hadrons, such as protons or neutrons, starting from the QCD action expression, which encodes fundamental quark and gluon interactions.

We classify the LQCD workflow in three steps. In the first step, hybrid-Monte Carlo updates are used to generate a statistical ensemble of gauge field configurations, which are saved in a chain of checkpoint files. Each of these files can be thought of as a snapshot of the quantum fluctuations of the gluon fields living in a discretized box of four-dimensional spacetime (three space dimensions plus time). For the purposes of the IO-SEA testing and benchmarking workflow, we refer to this phase as “Step A”.

These gauge configuration files are used as background data for calculations of physical quantities in subsequent steps. In fact, an ensemble of gauge configurations can be used for numerous physics calculations. Gauge field configuration ensembles have useful lifetimes of decades and are routinely shared between international collaborations.

In a run of Step B of the workflow, a gauge configuration is read in and used to define a large, sparse linear system. Solutions to this system are called “quark propagators” and are the building blocks of most LQCD physics calculations. We save them to disk as the solver produces them.

In Step C, we read (in groups) propagators produced on a given configuration and combine them together in “contractions”, similar to a scalar product. Contractions produce short arrays called hadron correlators, which we analyse offline.

In the IO-SEA benchmarking workflow we launch Step B and C in a single batch script.

The LQCD benchmarking workflow is illustrated in Figure 14.

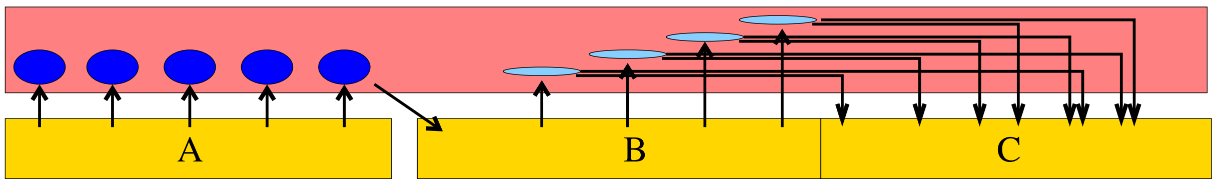


Figure 14: The LQCD workflow diagram shows the flow of major data files between the benchmark workflow steps *A*—*C* (in yellow) and storage (pink). The dark blue ovals represent the five checkpoint gauge field configurations files written out during the HMC step *A*. The last checkpoint is read in during Step *B* to produce four propagator files. Each of the four propagator files is read in twice during Step *C*. They are to be combined in different ways. Steps *B* and *C* are shown co-joined as they are launched from the same SLURM batch file with two successive `srun` calls.

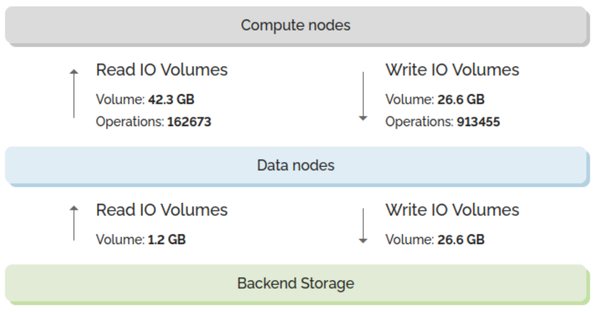


Figure 15: IOI report showing data reuse in the SBB with the LQCD workflow.

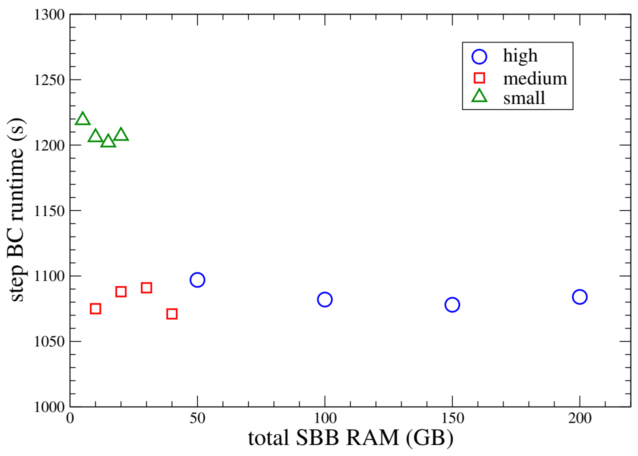


Figure 16: Runtime in seconds for the combined Step *BC*, versus total SBB RAM allocation.

8.2. LQCD in the IO-SEA environment

8.2.1. SBB service

In IO-SEA Deliverable 1.4 [1] we gave a detailed report of our experience using the SBB service in the LQCD workflow. In the tests performed for D1.4 we allocated the maximal amount of RAM for our SBB service session, that is in the SBB attributes we specified `flavor: high` (50 GB RAM per datanode) and four datanodes. This was chosen so that we could test the SBB service in optimal conditions.

We were curious about how the RAM allocation in the datanodes affects actual workflow performance, so we ran the LQCD workflow with every combination of SBB flavor and number of datanodes. For each, we extracted the runtime of the combined Step *BC*. .

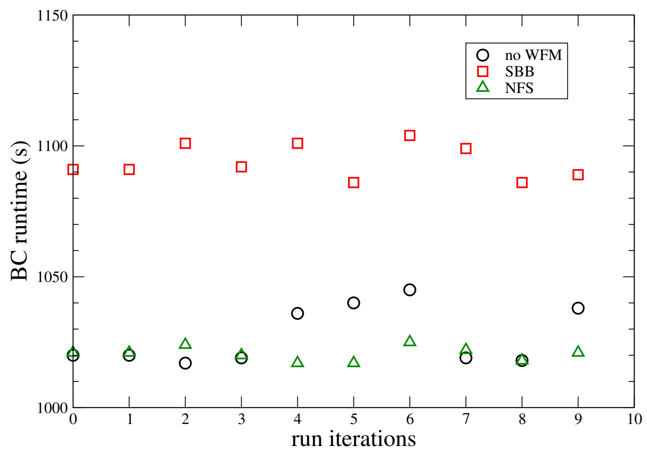


Figure 17: Ten iterations of a comparison test of LQCD benchmark Step *BC* runtime with the SBB service, the NFS service, and with direct I/O to the AFSM storage.

As described in [1], flavor `high` allocates 50GB RAM and 1TB of NVMe storage per datanode, `medium` provides 10GB of RAM and 500 GB of NVMe per datanode, and flavor `small` allocates 5GB of RAM per datanode with no NVMe storage.

We performed this test twice, once with the SBB target on the fast AFSM storage, and once with the target on the slower NFS-mounted project. Given the total write volume of the workflow is 26.6GB, as shown in Figure 15, one might expect that performance would degrade as the RAM size decreases below this number. Both tests showed the same result, though the effect is more pronounced in the case of the target directory on the NFS `/p/project/` storage. We show this in Figure 16. Performance remains approximately constant with the `high` and `medium` flavors down to 10GB of total RAM. However we see drastic performance deterioration with the `small` flavor even with four datanodes giving a total of 20GB of RAM.

It could be that the 5GB of RAM per datanode for the `small` flavor is too close to the 4.8GB file size of the propagators, alternately, or additionally, the performance degradation could be due to the lack of NVMe storage.

8.2.2. NFS Service

We have tested the LQCD benchmarking workflow with the NFS ephemeral service. Given our experience running the SBB service in the IO-SEA WFM, it was a straightforward adaptation of the WDF to run with the NFS service. We give a sample LQCD WDF for the NFS service in Listing 8.1.

In the context of this project we have used JUBE [8] to automate and standardize our LQCD benchmark execution, both with and without WFM services, as described in D1.4[1]. JUBE allowed us to run repeated iterations of the workflow comparing the effect of running with the SBB or the NFS service with direct I/O to and from the AFSM storage on DEEP. The results, shown in Figure 17, showed surprisingly good NFS service performance. Not only was the NFS service consistently faster than the SBB, it was also often faster than the direct AFSM access, which showed some slight performance slowdowns, perhaps when the network or storage module were particularly busy.

```
workflow:
  name: LQCD_WFM_ABC

services:
  - name: lqcd-nfs
    type: NFS
    attributes:
      namespace: HESTIA@/afsm/iosea/USE_CASES/LQCD/DATASETS//lqcd-bm-dataset-2024-01-12-122512-ID2
      mountpoint: /afsm/iosea/USE_CASES/LQCD/MNT/
      storagesize: 50GiB
      datanodes: 1
      location: dp-esb

steps:
  - name: step_A
    command: "sbatch /p/project/iosea/USE_CASES/LQCD/sub_deep-HMC-stepA.sh"
    services:
      - name: lqcd-nfs

  - name: steps_B_and_C
    command: "sbatch /p/project/iosea/USE_CASES/LQCD/sub_deep_meas_stepBC.sh"
    services:
      - name: lqcd-nfs
```

Listing 8.1: WDF `lqcd-NFS-wdf.yaml` for execution of LQCD workflow steps with the NFS service, with `storagesize` of 50GiB.

The NFS runs were our first experience using IO-SEA datasets. While we cannot yet reap the benefits of the HSM using the dataset to station our data in the storage hierarchy, we note that using datasets presented no great difficulties and we made only small tweaks to the initialization of the workflow. In the initial run of each workflow step, we set up the data directories using `mkdir` and `cp` commands. For the benchmarks we now prefix these with `srun`, giving these shell commands access to the dataset within the NFS service, e.g. :

```
srun -N 1 -n 1 mkdir -p $SAVE_DIR
srun -N 1 -n 1 cp -i ${initial_file} ${SAVE_DIR}/
```

In the case of the main workflow steps using a large number of compute nodes, this could be put into a new batch script for a "Setup" step with only a single node allocated to avoid idle resources.

With the deployment of WFM version 1.6.0, we have a new option: we could, from the login nodes, generate a template directory structure, complete with any starter data files and recursively copy it into a new dataset with the `iosea-migrate` command:

```
iosea-migrate -S 50GB ${TEMPLATE_DIRECTORY} HESTIA@/afsm/iosea/LQCD/my_lqcd_dataset.
```

We have tested the `iosea-migrate` command in this manner, but not yet incorporated it into the benchmarking workflow.

8.2.3. DASI service

The Chroma software system uses a specialized I/O library, QIO [14], which allows parallel I/O operations through offsets of the pointer to the binary file. Additionally, the program expects POSIX file paths to input and output data files. For these reasons, adapting the Chroma application to use DASI natively is difficult, if not impossible with the current DASI implementation.

Instead, we have tested using the DASI service to archive gauge field configuration files generated in the HMC stage (Step *A*) of the workflow.

In this example we use the DASI semantic keys shown in the schema in Listing 8.2. For the first level we use the physics parameters of the simulation. The first two keys `FermAct` and `GaugeAct` are strings which refer to the choice standard lattice *actions*, or the specific conventions used to discretize the quark and gluon fields, respectively. The coupling constant `beta` controls the lattice spacing, `volume` is a string representation of the lattice dimensions, and `mass` encodes a quark mass.

The next level keys include and `index` of the configuration stored in the file, and a file `type` to differentiate between the actual configuration file and an XML file of metadata which can be used to restart the HMC evolution from this configuration. The final key `dummy` would reference records inside a file, but this is not relevant for the binary gauge field files. The '?' indicates it is an optional parameter.

```
[ FermAct, GaugeAct, beta, volume, mass [ index, type [ dummy? ] ] ]
```

Listing 8.2: A sample DASI `schema` file for archiving gauge configuration files.

This schema file must be reference by a DASI configuration file in yaml format, such as listed in Listing 8.3. In addition to the path to the schema, the root path is a mount point for the DASI dataset.

```
schema: /p/project/iosea/USE_CASES/LQCD/DASI/schema
catalogue: toc
store: file
spaces:
  - roots:
    - path: /afsm/iosea/USE_CASES/LQCD/DASI_DATA
```

Listing 8.3: A sample DASI configuration file, which references the schema file in Listing 8.2.

For the initial DASI service tests we consider a workflow with two steps: an `archive` step which copies configuration files and their associated XML metadata file into a DASI-service dataset, and an

```
workflow:
  name: LQCD_DASI_TEST
services:
  - name: lqcd-dasi
    type: DASI
    attributes:
      dasiconfig: /p/project/iosea/LQCD/DASI/lqcd_dasi.yaml
      namespace: HESTIA@/afsm/iosea/USE_CASES/LQCD/DATASETS/DASI-{{ INDEX }}/
      storagesize: 50GiB
      datanodes: 1
      location: dp-esb
steps:
  - name: archive
    command: "sbatch /p/project/iosea/LQCD/archive_to_DASI.sh {{ START }} {{ STOP }} "
    services:
      - name: lqcd-dasi
  - name: extract
    command: "sbatch /p/project/iosea/extract_from_DASI.sh {{ DIR }} {{ START }} {{ STOP }} "
    services:
      - name: lqcd-dasi
```

Listing 8.4: WDF `lqcd-DASI-wdf.yaml` for using the DASI service to archive gauge configuration checkpoint files.

extraction step which can copy files from DASI storage to the disk. In Listing 8.4, we give an example of a WDF for this test workflow.

We take advantage of the WFM feature of passing variables to the WDF and workflow steps. With the command:

```
iosea-wf run -s dasitest-0 -t archive -d START=6 -d STOP=10
```

We launch a step to copy gauge configurations with index $6, ...10$ to a DASI dataset. To extract the same files from a DASI dataset to a destination directory we could use:

```
iosea-wf run -s dasitest-0 -t extract -d DIR=/afsm/LQCD/RESTORE-0/ -d START=6 -d STOP=10
```

The associated SLURM scripts for archiving to and extracting from a DASI dataset is given in Listing 8.5 and Listing 8.6, respectively.

8.3. Production workflow models

We consider which of these services or combination of services might be used in an LQCD workflow in a mature IO-SEA environment on an exascale system.

```
#!/bin/bash -x
#SBATCH --account=iosea
#SBATCH --nodes=1
#SBATCH --ntasks=1
#SBATCH --ntasks-per-node=1
#SBATCH --time=01:00:00
#SBATCH --partition=dp-esb
#SBATCH --gres=gpu:1
#SBATCH --job-name=archive

start=$1
stop=$2

# Define some variables here
beta=3.6
nx=32
nt=64
vol=l${nx}t${nt}
gauge_act=WILSON_GAUGEACT
ferm_act=CLOVER
mass=0.2

CONFIG_DIR=/afsm/iosea/LQCD/DATA//CONFIGS/L32T64/B3.6/

for (( i=$start; i<=$stop; i++ ))
do
    gauge_cfg_file=${CONFIG_DIR}/cfg_b3.6_l32t64_cfg_${i}.lime
    metadata=${CONFIG_DIR}/cfg_b3.6_l32t64_restart_${i}.xml

    config_keys="FermAct=${ferm_act},GaugeAct=$gauge_act,beta=${beta},\
        volume=${vol},index=${i},mass=${mass},type=config"
    restart_keys="FermAct=${ferm_act},GaugeAct=$gauge_act,beta=${beta},\
        volume=${vol},index=${i},mass=${mass},type=restart_xml"

    srun dasi-put --config=lqcd_dasi.yaml $config_keys $gauge_cfg_file
    srun dasi-put --config=lqcd_dasi.yaml $restart_keys $metadata
done
```

Listing 8.5: A sample SLURM script to run the `archive` step.

8.3.1. Dataset scheme

In such a system we would want to take advantage of IO-SEA datasets for efficiently managing our data within the hierarchical storage system. The first question is how might we group LQCD data into datasets. From our brief experience using the IO-SEA prototype and datasets with the NFS and DASI services, a likely arrangement would be a dataset containing all of the gauge field configurations generated in Step *A* for a given project, and a separate dataset for the propagator and correlator files produced in Steps *B* and *C*. The logic is that ensembles of gauge field configurations have useful lifetimes spanning a decade or more and many projects, and are often shared between collaborations, while the useful lifetime of propagators and correlators is usually just the length of a project.

A further question is whether to use the DASI service and storage interface. There are obvious advantages for data organization, but the drawback that the current LQCD applications cannot natively access DASI storage. A superb solution is the proposed idea of allowing multiple namespaces to expose the same dataset. In this case the data could be accessed as either POSIX files or as DASI objects as needed.

```bash
#!/bin/bash -x
#SBATCH --account=iosea
#SBATCH --nodes=1
#SBATCH --ntasks=1
#SBATCH --ntasks-per-node=1
#SBATCH --time=01:00:00
#SBATCH --partition=dp-esb
#SBATCH --gres=gpu:1
#SBATCH --job-name=extract

DEST_DIR=$1

start=$2
stop=$3

# Define some variables here --- could be passed in as arguments
beta=3.6
nx=32
nt=64
vol=l${nx}t${nt}
gauge_act=WILSON_GAUGEACT
ferm_act=CLOVER
mass=0.2

for (( i=$start; i<=$stop; i++ ))
do
    cfg_file=${DEST_DIR}/cfg_b${beta}_${vol}_cfg_${i}.lime
    metadata=${DEST_DIR}/cfg_b${beta}_${vol}_restart_${i}.xml

    # index i changes each iteration
    config_keys="FermAct=${ferm_act},GaugeAct=$gauge_act,beta=${beta},volume=${vol},index=${i},mass=${mass},type=config,dummy=dummy"
    restart_keys="FermAct=${ferm_act},GaugeAct=$gauge_act,beta=${beta},volume=${vol},index=${i},mass=${mass},type=restart_xml,dummy=dummy"

    srun dasi-get --config=lqcd_dasi.yaml $config_keys $cfg_file
    srun dasi-get --config=lqcd_dasi.yaml $restart_keys $metadata
done
```

Listing 8.6: A sample slurm script to run the `extract` step.

8.3.2. Service choices

One possible model that we have explored is to use either the SBB or NFS service to run the LQCD workflow, and then periodically run the `archive` step to save gauge field configurations into a DASI archive. This is illustrated in Listing 8.7.

We note that in the current version of the WFM, 1.6.0, we are not able to connect two ephemeral services to a workflow step. Therefore in our WDF in Listing 8.7 we use only the DASI service in the archive step. In the current implementation this is risky, as the archive step takes data from the background file system, rather than the SBB, and while the workflow session is active, it is not guaranteed that the disk has been synchronised to the SBB.

We have tested this workflow and found that in practice there was no problem. We ran the workflow including archiving gauge field configurations to DASI and extracting a new copy to disk. We found no checksum mismatches between the original and the extracted file.

The choice between SBB and NFS services probably depends on the problem size. In the medium size problem in the LQCD benchmarking workflow NFS beats SBB in performance, as seen in Subsection 8.2.2. We suspect that with large problem sizes requiring many nodes, the ability to do parallel read and write operations to a single gauge field configuration or propagator file becomes important. In these cases, serial I/O operations require a significant number of communication calls after the read to distribute data to the compute nodes, or before a write to collect data and send it to the I/O node.

```
workflow:
  name: LQCD_WFM_SBB-DASI
services:
  - name: lqcd-sbb1
    type: SBB
    attributes:
      targets: /afsm/iosea/USE_CASES/LQCD/DATA/JUBE_BENCHMARKING/WFM
      flavor: high
      datanodes: 4
      location: dp-esb

  - name: lqcd-dasi
    type: DASI
    attributes:
      dasiconfig: /p/project/iosea/LQCD/DASI/lqcd_dasi.yaml
      namespace: HESTIA@/afsm/iosea/LQCD/DATASETS/DASI-{{ INDEX }}/
      storagesize: 50GiB
      datanodes: 1
      location: dp-esb

steps:
  - name: step_A
    command: "sbatch /p/project/iosea/LQCD/sub_deep-HMC-TEMPLATE.sh {{ CLEANSTART }} "
    services:
      - name: lqcd-sbb1
  - name: steps_B_and_C
    command: "sbatch /p/project/iosea/LQCD/sub_deep_esb_spec_gpu_TEMPLATE-rsd.sh"
    services:
      - name: lqcd-sbb1
  - name: archive
    command: "sbatch /p/project/iosea/LQCD/archive_to_DASI.sh {{ START }} {{ STOP }} "
    services:
      - name: lqcd-dasi
  - name: extract
    command: "sbatch /p/project/iosea/LQCD/extract_from_DASI.sh {{ DIR }} {{ START }} {{ STOP }} "
    services:
      - name: lqcd-dasi
```

Listing 8.7: WDF `lqcd-SBB-DASI-wdf.yaml` for using the SBB service to run the LQCD workflow and the DASI service to archive gauge configuration checkpoint files.

If the SBB could be connected to the HESTIA backend storage, as proposed, we may be able to get the best of both worlds parallel I/O and direct access to a HESTIA dataset. Or perhaps the NFS service could be upgraded to a fully POSIX-compliant protocol, allowing many-nodes-to-one-file parallel I/O.

8.4. Conclusions

We have tested the LQCD workflow with the three available IO-SEA ephemeral storage services: SBB, NFS, and DASI. The SBB and NFS services perform similar functions of accelerating workflow I/O, while the DASI service can improve management of LQCD data. Each presents different advantages.

In our tests, we have found no barriers to running production LQCD workflows within the IO-SEA workflow manager, and the porting from the traditional batch submission was relatively painless, though the addition of creating job dependencies in the workflow manager would greatly relieve the human effort to keep a large project moving.

It would be very interesting to test some of the proposed, but not yet deployed features with the LQCD workflow.

Part I.

Summary

9. Lessons Learned and Recommendations

In the course of testing scientific workflows within the IO-SEA environment, we have been noting aspects that work well as well as features that could be improved in a mature deployed workflow manager and storage system.

We begin by noting that the scientific users have ingrained habits, often from years of interacting with HPC systems in a familiar way. Any change in this routine is a disruption before we uncover the advantages of the new paradigm.

Nonetheless, our charge as use case teams is to give feedback to the co-design process. In this chapter we describe lessons learned in this process and recommendations for the system.

9.1. WFM user interface

For the most part the features introduced in the current version (1.6.0) of the WFM work as expected. We list some features that would make the system easier to use.

- **Simple job cleanup:** In the current setup, SBB and other services can get stuck if requested resources are not available. We have not fully understood the failure modes and their causes, however services are stuck in states where they block no resources and also in states where they block resources. The latter is particularly an issue for the limited SBB RAM resources. In each case it is sometimes necessary that a system administrator cleans up the stuck services. It would be particularly useful to have a robust tool for the users to clean up their own services in any state.
- **Step arrays and dependencies:** Most HPC users use Slurm-type dependency rules between submitted jobs, increasing throughput and decreasing the need for constant monitoring of the batch queue. The ability to program such dependencies and job arrays for workflow steps with WFM commands would be valuable.
- **Useful error messages:** We have tested the IO-SEA Prototype system in many ways, and intentionally and unintentionally some of the access patterns have been beyond the foresight of our developer colleagues. We have discovered innumerable new failure modes. Our hope would be for a mature system to have helpful error messages for most of these cases.

9.2. IOI

The IOI monitoring framework has proved to be an extremely valuable tool for understanding the behavior of workflows run on the DEEP cluster within and without the IO-SEA environment. It is particularly useful for diagnosing problems which may not leave a signal in the application output or error logs, but which may nevertheless affect performance to some degree. Two real examples of user errors detected only with IOI during our testing runs are:

- **Misconfigured parallel I/O:** In this instance, an LQCD job was submitted with flags intended to select parallel I/O mode. However, as the IOI can show the number of process participating in I/O operations, it was apparent that a single node was still performing all of the I/O operations. The user was able to then find and correct a misconfigured input file to get true parallel I/O and improved performance.
- **Misconfigured SBB service:** During testing users occasionally discovered on IOI that the SBB was not intercepting I/O operations. The cause was usually uncovered as a misconfigured SBB target or a symbolic link in an I/O path.

In both of these cases, applications execute without error and produce correct results. However the missing functionality induces a performance cost, which would be significant were the workflows scaled up to production size. Both are easily diagnosed in IOI.

Some suggestions for the IOI tool:

- **Incorporate further ephemeral storage services:** Having seen the benefits of IOI in diagnosing the behavior of the SBB, we feel IOI could be useful in understanding the NFS or other dataset-based ephemeral services within a workflow.

9.3. LLView

We have found LLView [15] is also a useful tool, particularly for understanding how well hardware resources are utilized by the application. Given the focus on I/O and storage in the IO-SEA project, we have not utilized LLView as much as IOI.

We note that the LLView job listing now includes I/O reports drawn from IOI, and also includes links to the IOI report, both useful developments.

We have at this point a single suggestion for this improvement of the LLVies tool:

- **Closer binding between LLView and IOI:** Perhaps IOI and LLView could be hosted on the same server giving yet easier integration between the two services.

9.4. SBB service

- **SBB resource management** The SBB service is the easiest for a user to employ, but it relies on limited datanode RAM storage. As users launch their workflow steps by hand with the `iosea-wf run` command, there is often time when the SBB resources are idle between workflow steps, and occasionally forgetting to end the workflow session. This ties up idle RAM resources.

  Several solutions to this issue come to mind:

  – Institute a SBB timeout after some amount of elapsed time when no workflow steps are pending in the batch queue.

  – Rely on automated workflow step submission (see previous section).

  – Develop a dormant mode for SBB sessions that idle. The RAM image could be copied to a dataset in datanode storage and reloaded as a workflow step enters the SLURM "configure" state. Tighter coordination between the Slurm scheduler and the datanodes could make this seamless for the user.

9.5. NFS service

The NFS service has surprised us with its performance, being faster than either the SBB service or direct I/O to the AFSM storage for medium sized workflows with data re-use, such as the LQCD benchmark workflow.

A few notes

- **Parallel I/O:** As the NFS protocol is not fully POSIX compliant some parallel I/O operations are not available, for example multiple nodes writing to the same file. The lack of this functionality will cause delays with larger simulations and more nodes, as serial read and writes induce a lot of communication to gather or distribute data.
- **Command line tools:** A suite of command-line tools to move data between the datasets would be useful. For example, the current `iosea-migrate` tool is useful to initialize a dataset with data from a POSIX filesystem. A corresponding tool to extract data could be useful for quickly checking workflow progress.

9.6. DASI service

We found the DASI service to be useful to most use-case workflows. In its current implementation the syntax for the DASI ephemeral service API is slightly redundant and clarity and usability could be improved:

- **WDF format stream-lining:** In the DASI ephemereral service declaration in the WDF, one must specify a DASI configuration file in the `dasiconfig` attribute. (See, e.g., Listing 8.4). This file in turn refers to the file containing the relevant DASI schema. However, this same DASI configuration file must be specified as an argument from the CLI `dasi-put` and `dasi-get` commands:

  ```
  dasi-put --config=${configfile} $config_keys $file
  ```

  Useful behavior would be for the activation of the DASI ephemeral service to define an environment variable referring to the DASI configuration file, which would be the default for the `dasi-put` and `dasi-get` commands.

9.7. Datasets

- **Dataset management:** In version 1.2, datasets are POSIX files located in a user provided directory. As such, they can be managed as regular files (listed, moved, renamed, deleted,...) so we did not develop any specific tools.

  In version 1.4, datasets are still visible as POSIX files in a user directory, but as data are now stored in HESTIA, the files contain only the HESTIA objects IDs. Users can still list, rename or move their datasets. However, deleting a dataset became a bit more complex, as deleting the POSIX file itself is no longer sufficient. For time reason, no "delete" command has been developed, but it has been considered as acceptable, as not a lot of datasets would be created on the IO-SEA Pilot system.

- **Dataset naming:** In version 1.6, We chose to apply the DASI objective of hiding storage format details to users, implementing a very simple Ephemeral Service description based upon the `dasi-config.yaml` file (already prepared for the application). We did not ask the user for a dataset name, and instead created it from information in the dasi-config file. This led to dataset names not being meaningful for users, as for instance "`2744395ba1e5a548be8e396d39679b0a6d2cd4b51b7b8120652c95a09354a533`".
  It would be more coherent with previous versions and simpler if the user can decide themself the name of the file representing its dataset.

- **Large datasets:** The problem experienced by the RAMSES use case with large NFS datasets is related to internal tools used to migrate datasets to long-term storage in HESTIA. This should no longer be an issue in WFM v2.X when each file is a HESTIA object, rather than the entire dataset.

10. Summary

In this final report of IO-SEA Work Package 1 we have reported on our experience testing the IO-SEA WFM and ephemeral storage services as they are deployed in early 2024. There has been rapid development of the available features in the recent months and weeks as the IO-SEA results of three years of the IO-SEA project efforts became available to users.

The WFM and ephemeral services present a novel way of managing HPC workflow steps and the resulting data. We have tested all IO-SEA use cases with the final deployed version of the IO-SEA WFM and describe the results in Chapter 9. We have seen that our five scientific use cases with their diverse workflows can adapt to this paradigm. We have described cases, such as workflows with a large amount of data re-use, where the SBB and NFS services already provide a performance boost without any code adaptation.

Additionally we have explored how data might be organized with DASI semantic keys, either by using the *C++* or *Python* API within the application, or by using DASI commands from the command line.

From this testing experience we have noted a number of features that we felt could be changed or added to make the IO-SEA storage ecosystem more useful over a wider range of workflows and data access patterns. We have noted many of these in Chapter 9.

This experience of running scientific use case applications in the IO-SEA environment has been instructive. Whereas previously we might have considered the placement of individual HPC jobs on the appropriate compute hardware as paramount to optimization, the project has highlighted the need to view entire workflows, including also the flow of data between workflow steps. We are introduced to datasets as a concept useful in managing workflow data from creation in the application through its lifetime in the tiers of hierarchical storage.

We conclude by noting that, in this short testing period since the WFM has been deployed, we have scratched the surface in exploring the ways to use this IO-SEA environment. We wish we had more time for more in-depth testing, as there are many experiments we would like to do. We feel it would be exciting and rewarding to see and use the concepts developed in this project on a production machine in the future.

List of Acronyms and Abbreviations

A

AFSM	The All-Flash Storage Module is a purely flash-based storage module of the DEEP system.
AMR	Adaptive Mesh Refinement.

C

CEA	The French Alternative Energies and Atomic Energy Commission.
CLI	Command Line Interface.
CLM	Community Land Model.
COSMO	Consortium for Small-scale Modeling.
CUDA	The Compute Unified Device Architecture is a parallel computing platform as well as an API that allows for the communication with certain types of graphics-processing units.

D

DASI	Data Access and Storage Interface developed in Work Package 5.
DEEP	The Dynamical Exascale Entry Platform (DEEP) is a multi-component project project to prepare for upcoming exascale HPC systems. The DEEP-SEA project is latest component of this project. It is also used to refer to the prototype developed in the DEEP projects based on context..

E

ECMWF	European Centre for Medium-Range Weather Forecasts.

F

FDB	Fields DataBase, the ECMWF meteorological object store.
FZJ	Forschungszentrum Jülich, in Jülich, Germany, is one of the largest research centres in Europe and a member of the Helmholtz Association.

H

Hercule	Parallel I/O and data management library developed at CEA.
HESTIA	Hestia (Hierarchical Storage Tiers Interface for Applications) is a Hierarchical Storage Management (HSM) wrapper for Object Stores. It is being developed in

	the context of the IO-SEA project for Exascale Storage I/O and Data Management. .
HMC	The Hamiltonian Monte Carlo algorithm (originally known as hybrid Monte Carlo) is a Markov chain Monte Carlo method for obtaining a sequence of random samples which converge to being distributed according to a target probability distribution for which direct sampling is difficult.
HPC	High-Performance Computing.
HSM	Hierarchical Storage Management.
I	
IOI	The IO Instrumentation tool collects I/O-related metrics and provides diagnostic information about workflow read and wrinte operations.
IT4I	IT4Innovations National Supercomputing Centre at VSB Technical University of Ostrava, Czech Republic.
J	
JUBE	The JÜlich Benchmarking Environment is a script based framework to easily create benchmark sets, run those sets on different computer systems and to evaluate the results.
L	
LQCD	Lattice quantum-chromodynamics is a numerical framework for calculating physical properties of hadrons, composite particles composed of quarks.
M	
mmap	mmap is a POSIX-compliant Unix system call that maps files or devices into memory.
MPI	The Message Passing Interface is a common API for communication between tasks running on one or more computers.
MSA	Modular Supercomputing Architecture.
N	
NCL	The NCAR Command Language is an open source interpreted language, designed specifically for scientific data processing and visualization.
NetCDF	NETwork Common Data Form is a community standard, machine-independent data format that support the creation, access, and sharing of array-oriented scientific data. It is extensively used in Earth system modelling.
NFS	Network File System, a file system allowing to share files between many nodes over a TCP/IP network.

NVMe	Non-Volatile Memory Express.
P	
ParFlow	A physically-based and spatially distributed hydrological model solving surface and subsurface flows in a massively parallel computational framework.
PFB	ParFlow Binary format is a binary file format which is used to store ParFlow grid data.
POSIX	Portable Operating System Interface is a family of standards specified by the IEEE Computer Society for maintaining compatibility between operating systems.
R	
RAMSES	Code to model astrophysical systems, featuring self-gravitating, magnetised, compressible, radiative fluid flow, using AMR technique..
S	
SBB	Smart Burst Buffer is a hardware accelerator that can be included in the I/O data path and accelerate I/O on specific files for targeted applications. Offered as an IO-SEA ephemeral service..
SLURM	SLURM is an open-source cluster-management and job-scheduling system.
T	
TSMP	Terrestrial System Modelling Platform is an open source scale-consistent, highly modular, massively parallel regional Earth system model.
W	
WDF	The workflow description file is a YAML configuration file describing ephemeral services and steps invoked in a workflow.
WFM	The IO-SEA Workflow Manager starts ephemeral storage services, and runs workflow steps in the IO-SEA storage environment. .

Bibliography

[1] P. Couvee E. B. Gregory. IO-SEA D1.4: First application port and evaluation. Technical report, IO-Software for Exascale Architectures, Sep 2023.

[2] P. Couvee, S. Happ, and M. Rauh. IO-SEA D2.3: Ephemeral Data Access Environment - Final Version. Technical report, IO-Software for Exascale Architectures, Feb 2024.

[3] Sébastien Gougeaud, Guillaume Courrier, Yoann Valeri, Buket Benek Gursoy, Venkatesh Kannan, Philippe Couvée, Simon Derr, James Grogan, and Katie O'Connor. IO-SEA D4.3: Hierarchical Storage Management Feature: Final version of the validations. Technical report, IO-Software for Exascale Architectures, Mar 2024.

[4] D. Medieros and S. Markidis. IO-SEA D5.6: Final application workflow adaptations to the DASI. Technical report, IO-Software for Exascale Architectures, 2024.

[5] M. E. Holicki, E. B. Gregory, and M. Golasowski. IO-SEA D1.2: Application use cases and traces. Technical report, IO-Software for Exascale Architectures, Dec 2021.

[6] Loic Strafella and Damien Chapon. Boosting I/O and visualization for exascale era using Hercule: test case on RAMSES. *CoRR*, abs/2006.02759, 2020.

[7] Bressand, Colombet, Fontaine, and Harel. Hercule: a library of scientific data management for numerical simulation. *CHOCS*, 41, 29-37, 2012.

[8] Jülich Supercomputing Centre. JUBE — Jülich Benchmarking Environment. http://www.fz-juelich.de/jsc/jube, 2008.

[9] mmap documentation. `https://man7.org/linux/man-pages/man2/mmap.2.html`.

[10] D. Medeiros and S. Markidis. IO-SEA D5.3: Adaptations of Application Workflows to DASI. Technical report, IO-Software for Exascale Architectures, Mar 2023.

[11] J. Wong, M. Cakircali, and J. Hawkes. IO-SEA D5.4: Demonstrations of data-centric workflows using DASI for in-situ processing and visualisation. Technical report, IO-Software for Exascale Architectures, 2024.

[12] ECMWF. FDB. https://github.com/ecmwf/fdb, 2024.

[13] ECMWF. Kronos, HPC workload simulator. https://github.com/ecmwf/kronos, 2018.

[14] QIO, the QCD Input/Output applications programmer interface. http://usqcd-software.github.io/qio/.

[15] Jülich Supercomputing Centre. LLview — Graphical monitoring of batch system controlled cluster. http://www.fz-juelich.de/jsc/llview, 2021.