EuroHPC-01-2019

IO-SEA

IO – Software for Exascale Architectures

Grant Agreement Number: 955811

D2.3

Ephemeral Data Access Environment - Final Version

Final

Version:	2.0
Author(s):	P. Couvée(EVIDEN), S. Happ(ParTec), M. Rauh(ParTec)
Contributor(s):	S. Krempel(ParTec), N. Derbey (EVIDEN), L. Bozga(EVIDEN), M. Golasowski(IT4I)
Date:	February 28, 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:	D2.3
	Deliverable Nature:	Report
	Deliverable Level: PU *	**Contractual Date of Delivery:** 29 / February / 2024
		Actual Date of Delivery: 29 / February / 2024
	EC Project Officer:	René 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:** Ephemeral Data Access Environment - Final Version	
	ID: D2.3	
	Version: 2.0	**Status:** Final
	Available at: https://www.iosea-project.eu/	
	Software Tool: LaTeX	
	File(s): IO_SEA_D1.2.pdf	
Authorship	**Written by:**	P. Couvée(EVIDEN), S. Happ(ParTec), M. Rauh(ParTec)
	Contributors:	S. Krempel(ParTec), N. Derbey (EVIDEN), L. Bozga(EVIDEN), M. Golasowski(IT4I)
	Reviewed by:	J. Wong (ECMWF) J.O. Mirus (FZJ)
	Approved by:	Exec Board/WP7 Core Group

Document Status Sheet

Version	Date	Status	Comments
0.1	10.01.2024	Outline approved	
0.9	19.02.2024	Draft ready for internal review	
1.0	22.02.2022	1st internal review complete	
1.1	27.02.2024	post-1st-review edits complete	
2.0	28.02.2022	final draft ready for EU submission	

Document Keywords

Keywords:	IO-SEA, HPC, Exascale, Software

Contents

List of Figures

Executive Summary

This work package 2 final deliverable describes the IO-SEA runtime environment developed as part of the IO-SEA project. Its main components are on the login nodes side the **Workflow Manager** composed of the Command Line Interface, the API Server, the Resource Manager and the Orchestrator, and on the datanodes side the **Ephemeral Services** and the **ParaStation HealthChecker** monitoring engine. These components are integrated with ParTec's process manager **ParaStation Management** on the compute nodes side.

These elements have been integrated and also deployed on the IO-SEA Pilot system hosted at FZJ. This allows to expose the IO-SEA software stack to the use cases of work package 1.

In the first part of the document, we describe the run time environment and how its features have evolved from the first Version 1.0 until the now final version V1.6. On top of this, we share feedback from the use cases and also share some lessons learned.

In the second part of the document, we report the outcome of the studies on the Non-Volatile Memory technology and its potential use in the IO-SEA architecture. Due to changes on Seagate's business strategy, the scope of this work has been adapted by a recent amendment. We here report on the effort provided for this task 2.2.

1. Introduction

In part 1 of this last WP2 deliverable, we describe the IO-SEA run time components that have been developed, deployed on the IO-SEA Pilot system and shared with the WP1 use cases. The main components of this runtime environment are the Workflow Manager itself (up to and including its Version V1.6), containing the Command Line Interface, the API Server, the Resource Manager and the Orchestrator, and on the datanodes side the Ephemeral Services and the ParaStation HealthChecker monitoring engine. These components are integrated with ParaStation Management—the process manager of ParTec that is used alongside the Slurm scheduler on the supercomputing systems of FZJ including the IO-SEA Pilot system.

We describe also other components developed and tested outside of the IO-SEA Pilot system. These developments have not been fully deployed on the IO-SEA Pilot system and have thus not been exhaustively tested by the use cases. Then, we analyse the feedback received during the collaboration to optimize the use cases and share the lessons learned.

In the second part, we present the outcome of the studies on the Non-Volatile Memory technologies and their usage in the IO-SEA architecture. This second part has been deeply affected by the Seagate withdrawal of their participation in the project, as they were the task leader. In consequence, the studies have focused on the system integration of the Non-Volatile Memory on the datanodes.

Part I.

IO-SEA Run Time Environment

2. Workflow Manager

In deliverable 2.2, we describe the detailed architecture, the command line interface and the Workflow Description File, as well as the very first version of the IO-SEA run time environment set up on the virtual system at IT4I. We also described the planned development steps to add progressively the main IO-SEA features.

The following feature versions have been developed, deployed on the IO-SEA Pilot system and tested by the WP1 use cases:

- **Version 1.0 of the Workflow Manager**. This version introduced the support of the first Ephemeral Service (Smart Burst Buffer) on real datanodes.
- **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.
- **Version 1.6 of the Workflow Manager**. This version introduces a first connection to WP5 with the support of a new DASI ephemeral service.

Further versions of the Workflow Manager were planned and have been partially developed and tested, but not deployed on the IO-SEA Pilot system:

- **NFS Ephemeral Service storing each file as a Hestia object**, enabling individual file movement in the storage hierarchy.
- **Ephemeral Services Orchestrator**. It runs as a separate service receiving requests from the API server, handles the sessions instances, driving more complex processing such as data movers before and/or after running a step.

Each version is presented in the next sections of this chapter.

2.1. Workflow Manager V1.0

The main objective of the first version of the Workflow Manager was to introduce the command line interface to manage *sessions*. Indeed, Ephemeral Services are only available within a session created and deleted interactively by users. In this version, only one Ephemeral Service is proposed (the Smart Burst Buffer: SBB) and there is no support for datasets (as SBB does not require them).

A How-to document was prepared for WP1 use cases, providing a step-by-step recipe for usage. Here is a summary of this document:

First step is to prepare the Workflow Description File, as introduced in deliverable D2.2 [1]. Due to the IO-SEA Pilot system settings, the WDF must contain an additional “location” attribute in the “services”

part, as there is no default Slurm partition on the IO-SEA Pilot system. The location is the partition the ephemeral services creation and deletion jobs will be executed on. For instance :

```
workflow:
   name: Sample-Workflow

services:
   - name: sbb1
     type: SBB
     attributes:
       targets: /p/home/jusers/couvee1/deep/
       flavor: small
       datanodes: 2
       location: dp-cn

steps:
   - name: step_1
     command: "sbatch script2.sh"
     services:
       - name: sbb1

   - name: step_2
     command: "sbatch script1.sh"
     services:
       - name: sbb1
```

The IO-SEA Pilot system has been deployed with 4 data nodes, each hosting 2 NVMe disks. Users select the number of desired data nodes through the datanodes attribute.

We have defined 3 flavors, as required by the SBB:

- Flavor=small MemorySize=5GB Cores=2
- Flavor=medium MemorySize=10GB Cores=4 StorageSize=500GB
- Flavor=high MemorySize=50GB Cores=8 StorageSize=1TB

The combination of data nodes count and flavor parameters provides a large set of possibilities for datanode resources.

In order to simplify the system integration and avoid complex user authentication and identity management, each user has to launch its own API server in its environment. As there is no "Orchestrator" in this version, the API component is directly in charge of submitting the requests to Slurm. Since it operates using user credentials, there is no need for additional identity management features. To launch the API server, users simply launch the following command:

```
wfm-api&
```

Each user has also a private database where the API server stores the states of the various sessions. It is automatically created by the API server if necessary. The database file is named $HOME/.wfm-api.db.

The CLI has been deployed as a Python virtual environment in a shared directory. Users were invited to configure their environment to facilitate the use of the commands, for instance adding in their .bashrc file the following 2 lines:

```
alias wfm-api=/p/scratch/iosea/wfm/wfm-1.0.0/iosea_wfm_venv/bin/wfm-api
alias iosea-wf=/p/scratch/iosea/wfm/wfm-1.0.0/iosea_wfm_venv/bin/iosea-wf
```

Then, users were ready to start sessions and submit workflow steps. An example of such a session has been provided:

```
[couvee1@deepv ~]$ iosea-wf start -w sample-wdf.yaml -s test5
Check session status before starting any step:
[couvee1@deepv ~]$ iosea-wf status -s test5
NAME STATUS
test5 active
When your session is active, steps can be launched:
[couvee1@deepv ~]$ iosea-wf run -s test5 -t step_1
Check the progress of your steps until they are stopped:
[couvee1@deepv ~]$ iosea-wf status -s test5 -T
ID STATUS JOBID
1 stopped 359757
[couvee1@deepv ~]$
[couvee1@deepv ~]$ iosea-wf run -s test5 -t step_2
[couvee1@deepv ~]$ iosea-wf status -s test5 -T
ID STATUS JOBID
1 stopped 359757
2 starting 359758
[couvee1@deepv ~]$ iosea-wf status -s test5 -T
ID STATUS JOBID
1 stopped 359757
2 running 359758
[couvee1@deepv ~]$ iosea-wf status -s test5 -T
ID STATUS JOBID
1 stopped 359757
2 stopping 359758
[couvee1@deepv ~]$ iosea-wf status -s test5 -T
ID STATUS JOBID
1 stopped 359757
2 stopped 359758
[couvee1@deepv ~]$
When done with your session, just stop it to release datanodes resources:
[couvee1@deepv ~]$ iosea-wf stop -s test5
srun: ioi-slurm-manager: Bull IO Instrumentation enabled upon user request dp-cn28
Successfully stopped session test5
[couvee1@deepv ~]$
```

The Workflow Manager and its SBB Ephemeral Service have been used by almost all use cases successfully, and the results of their tests have been reported in the deliverable D1.5.

2.2. Workflow Manager V1.2

Version 1.2 of the Workflow Manager added several functionalities, the most interesting being:

- The support of a new ephemeral service, enabling the support of datasets that are stored as POSIX files.
- The support of a new command in the CLI that enables access to these newly supported datasets.

2.2.1. Datasets support

Datasets are supported through a newly supported Ephemeral Service called **NFS** that enables access to a Ganesha NFS server, configured to export a POSIX filesystem. This filesystem is stored itself as one (big) POSIX file whose name is the name of a dataset listed in the Workflow Description File.

Similarly to V1.0 of the Workflow Manager, a How-to document was written for WP1 use cases, describing the way this new kind of service should be specified in their Workflow Description File (from a user point of view, the main difference that comes with the new **NFS** Ephemeral Service resides in the way they describe it in the WDF):

```
workflow:
   name: Sample-Ganesha-Workflow

services:
   - name: nfs1
     type: NFS
     attributes:
       namespace: /afsm/iosea/couvee1/datasets/my-dataset-6GB
       mountpoint: /afsm/iosea/couvee1/mnt
       storagesize: 6GiB
       datanodes: 1
       location: dp-cn

steps:
   - name: step_1
     command: "sbatch script1.sh"
     services:
       - name: nfs1

   - name: step_2
     command: "sbatch script2.sh"
     services:
       - name: nfs1
```

Note in the example above the **namespace** attribute that specifies the name of a dataset: this is the name of the file where the POSIX filesystem exported by the Ganesha NFS server is stored. If the file does not exist when starting a session, a dataset will be created by the workflow manager as soon as that session is stopped. If the file exists, the corresponding dataset will be accessed. The

attribute has been named "namespace" and not "dataset" as in the IO-SEA architecture, a dataset could have more than one namespace. In future versions, the attribute format will evolve to become "dataset.namespace".

This choice has been made to simplify dataset management for the user: each dataset is a file stored in a user-provided directory. Using standard UNIX commands such as ls, rm, mv, ... users can manage their datasets.

The **mountpoint** attribute is the dataset mount point on the compute nodes during the steps execution.

2.2.2. The "iosea-wf access" command

This command provides the user with the instructions needed to access the specified session. Thanks to this command, the user is able to access the datasets the ephemeral services are configured for when the command is invoked. Thus, the user can, for example, interactively review or even update the dataset's content before proceeding to the next step.

We show below an excerpt from the How-to document, showing how the **iosea-wf access** command can be used to help launch an interactive environment in the context of an NFS Ephemeral Service in order to first check that the dataset is correctly mounted, then to populate that dataset before running a step in the current session (called *ganesha6G*). We show also that when the session is stopped, the namespace gets created.

```
[couvee1@deepv ~\]$ ll /afsm/iosea/couvee1/datasets
total 0
[couvee1@deepv ~]$ iosea-wf start -w ganesha-wdf.yaml -s ganesha6G
Check session ganesha4G status before starting any step
[couvee1@deepv ~]$ iosea-wf access -s ganesha6G
Type the following command in order to get access to session ganesha4G:
     /usr/bin/srun -J interactive -p dp-cn -N 1 -n 1 --bb "#BB_LUA GBF use_persistent
         Name=couvee1-ganesha6G-nfs1" --pty bash
Then type ^C to exit
[couvee1@deepv ~]$ /usr/bin/srun -J interactive -p dp-cn -N 1 -n 1 --bb "#BB_LUA GBF
    use_persistent Name=couvee1-ganesha6G-nfs1" --pty bash
srun: job 451830 queued and waiting for resources
srun: job 451830 has been allocated resources
[couvee1@dp-cn02 ~]$ df
Filesystem 1K-blocks Used Available Use% Mounted on
devtmpfs 98071468 0 98071468 0% /dev
. . .
10.2.19.212:/ganesha_58577 4175872 36864 4139008 1% /afsm/iosea/couvee1/mnt
[couvee1@dp-cn02 ~]$ cd /afsm/iosea/couvee1/mnt
[couvee1@dp-cn02 mnt]$ ll
total 0
[couvee1@dp-cn02 mnt]$
[couvee1@dp-cn02 mnt]$ mkdir input
[couvee1@dp-cn02 mnt]$ cp ~/toto.txt* input
[couvee1@dp-cn02 mnt]$ ll
total 0
drwxr-xr-x 2 couvee1 jusers 50 Nov 17 16:49 input
```

```
[couvee1@dp-cn02 mnt]$ exit
[couvee1@deepv ~]$ iosea-wf run -s ganesha6G -t step_1
Successfully submitted step_1 step: couvee1-ganesha6G-step_1_1
[couvee1@deepv ~]$ iosea-wf stop -s ganesha6G
Clean (iosea-wf status) the stopped session ganesha6G before reusing its name
[couvee1@deepv ~]$ ll /afsm/iosea/couvee1/datasets/
total 8852023
-rw-r--r--. 1 couvee1 iosea 6443499520 Nov 22 17:40 my-dataset-6GB
```

Please note, as indicated in the listing above, that each time a step is run, it is associated with an instance name composed of the concatenation of the user identifier, the session name, the step name, and an instance identifier. This method ensures the uniqueness of each step instance, thereby allowing, for example, the execution of a single step multiple times within a single session.

2.3. Workflow Manager V1.4

The major changes introduced in version 1.4 were:

- Support for heterogeneous slurm jobs, that was needed by TSMP use case in WP1 and was missing in the previous versions.
- Access to the NFS Ephemeral Service enabled via the Hestia server (in addition access via the the Ganesha NFS server) has been introduced in v1.2. With this feature, the HESTIA API has been called to store the datasets into the hierarchical storage. The references to the HESTIA objects are stored inside a POSIX file whose name is the name of a dataset listed in the Workflow Description File.

Again, a How-to document was provided to WP1 use cases, describing how this new service should be specified in their Workflow Description Files. The specification is exactly the same as the one for the NFS service, except that the namespace attribute can now be prefixed by the "**HESTIA@**" string, denoting that HESTIA special processing should be achieved:

```
workflow:
   name: Sample-Ganesha-Workflow

services:
   - name: nfs1
     type: NFS
     attributes:
       namespace: HESTIA@/afsm/iosea/couvee1/datasets/my-dataset-6GB
       mountpoint: /afsm/iosea/couvee1/mnt
       storagesize: 6GiB
       datanodes: 1
       location: dp-cn

steps:
   - name: step_1
     command: "sbatch script1.sh"
     services:
       - name: nfs1
```

```
  - name: step_2
    command: "sbatch script2.sh"
    services:
      - name: nfs1
```

Note above the **namespace** attribute contains the **HESTIA@** prefix, denoting the access to a HESTIA backend server. Similarly to what was presented in the previous chapter, the file named in that attribute will be used by the workflow manager:

- either to create a new dataset after a session is stopped,
- or to use an existing dataset: the data is retrieved at session start and stored back at session stop.

Since the principle is the same as that of the NFS service described in the previous chapter, we will not detail it here again.

2.3.1. Integration with the Resource Manager

Another improvement in version 1.4 was a first step towards the integration with the Resource Manager part of the Orchestrator that was under development (see Section 2.6).
When the Resource Manager is activated, allocating resources on behalf of an Ephemeral Service becomes a two-step operation:

- the Resource Manager is first asked by the Workflow Manager to reserve the needed resources.
- upon successful completion of the previous step, the burst buffer plugin is asked to actually start the Ephemeral Service in a second step.

In order for this feature to be supported, the burst buffer plugin had to be updated (note that this feature has only been implemented for the plugin related to SBB ephemeral service): the burst buffer plugin used to be in charge of both reserving and allocating itself any needed resource. It has been improved to allow 1. not doing the reservation anymore, 2. delegating the allocation part, on demand, to an "external allocator" (namely the Resource Manager).

2.3.2. Data migration progress

Any session that needs some data to be migrated between a POSIX filesystem to an IO-SEA dataset can call the migration tool as described in deliverable D4.3. The WP4 migration tool relies on a new command introduced in version 1.4: **iosea-wf update**. It is called by the migration tool each time it needs to report progress about data migration. This reported progress is a free-format string that is recorded for any step inside the Workflow Manager Steps-related database. This progress string is then outputted, if present, when the user asks for a step status, adding yet another connection to WP4.

2.4. Workflow Manager V1.6

Version 1.6 of the Workflow Manager added a new Ephemeral Service for DASI (Data Access and Storage Interface) backend. The DASI component itself is described in deliverables D5.1 and D5.5. The DASI configuration provides paths to datastore and index backend. A Ganesha NFS server is associated with each such path. It enables applications using DASI to use the workflow manager and to benefit from the backend management as an ephemeral service.

As for previous versions, a How-to document was provided to WP1 use cases, describing how this new kind of service should be specified in the Workflow Description Files.

```
workflow:
   name: Sample-DASI-Workflow

services:
   - name: dasi1
     type: DASI
     attributes:
       dasiconfig: /p/home/jusers/couvee1/deep/dasi-config/dasi-config.yaml
       namespace: HESTIA@/afsm/iosea/couvee1/datasets/
       storagesize: 2GiB
       datanodes: 1
       location: dp-cn

steps:
   - name: step_1
     command: "sbatch script1.sh"
     services:
       - name: dasi1

   - name: step_2
     command: "sbatch script2.sh"
     services:
       - name: dasi1
```

The specification for the DASI Ephemeral Service is similar to the NFS with two differences. Firstly, in order to identify the backend path a DASI configuration file is required. In this file, the backend path is specified by the DASI roots. Each such root is an absolute path used as a mountpoint on compute nodes during the step execution. Secondly, a repository name is expected instead of the file name for the namespace. This repository is used to store one dataset for each root described in the DASI configuration file. The first time a session is started with a new DASI configuration file, datasets are created, for the next run the same datasets are accessed. The "**HESTIA@**" prefix may be used for the namespace to specify HSM storage.

Note that this first implementation supports only one root path being defined in the DASI configuration file.

Even though three kinds of ephemeral services are now supported by the Workflow Manager, a limitation remains regarding the number of ephemeral services a given step is able to use: any step can use only a single Ephemeral Service at a time, preventing for example to define a step that would take as input a set of output files produced by earlier steps using SBB and produce itself some output

files into a HESTIA server. Similarly, it is not possible today to move datasets between two distinct services. Supporting multiple ephemeral services in parallel is a feature planned in version 2.0.

2.5. File storage as Hestia objects

With the ephemeral services integrated in versions up to 1.6, a dataset is stored as one big POSIX file or Hestia object, preventing the possibility to move individual user files in the storage hierarchy, and so preventing the implementation of data movers (feature of the IO-SEA stack allowing to move or copy files in a specific layer of the tiered storage environment, introduced in D2.2).

The next step in terms of ephemeral services is to enable the long term storage of each individual file as a Hestia object, that can be moved independently from the others in the same dataset. This is the objective of the integration of the Ganesha/KVSNS/GRH solution as an ephemeral service. The description of this integration has been done in the deliverable D5.2 and further explained in D5.6, chapter 3. In our case, launching the Ephemeral Service means setting up a storage volume for the cache and another storage volume in which we instantiate the Redis Key-Value store representing the POSIX namespace of the dataset.

The Workflow Manager handles the Ganesha/KVSNS/GRH Ephemeral Service similarly to the others, creating a file in the user-provided directory to materialize the dataset. This file contains the Hestia object ID of the Redis Key-Value store data.

The Workflow Description File for this service would look like:

```
workflow:
   name: Sample-KVSNS-Workflow

services:
   - name: NFS1
     type: NFSKVSNS
     attributes:
       namespace: HESTIA@/afsm/iosea/couvee1/datasets/my-kvsns-dataset
       mountpoint: /afsm/iosea/couvee1/mnt
       storagesize: 2GiB
       datanodes: 1
       location: dp-cn

steps:
...
```

The file "my-kvsns-dataset" contains the Hestia object ID of the Redis Key-Value store database. The size of the data cache volume is taken from the Workflow Description File each time a session is started, allowing to dynamically adjust it in sessions if needed.

This Ephemeral Service has been partially integrated on the IO-SEA Pilot system, but not exposed to users.

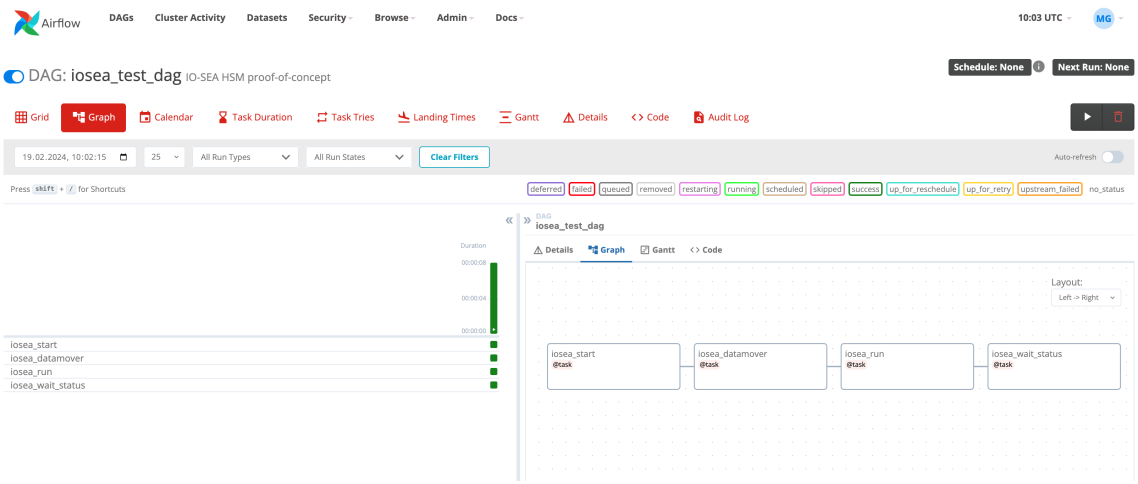

Figure 1: Proof-of-concept IO-SEA integration in Apache Airflow

2.6. Orchestrator and Data movers

The Orchestrator is a component of the Workflow Manager along with the Session Manager (SM) and Slurm API and provides an interface for the Session Manager allowing to execute workflow tasks on the cluster. It also includes the datanodes resource manager used by the Session Manager to reserve Ephemeral Services for workflows but also used by Ephemeral Services to attach to the resources reserved for starting the service itself. The Orchestrator is completely session-agnostic. In addition to the Resource Manager (RM) endpoints (which provide an interface to manage datanode resources), the Orchestrator API provides endpoints to manage ephemeral services and workflow steps.

A session is started by reserving resources for all ephemeral services that are described in the services section of the workflow description file. In the first implementation the Ephemeral Services (ES) are started just after reservation. In a second phase it is planned to implement a lazy policy in order to start the ES and, thus, use the datanode resources as late as possible (just before the service is used). At the end of the session all ESs will be released. Another possibility is to give the user the possibility of stopping an ES explicitly in a dedicated step. The resource manager API has been implemented by IT4I in cooperation with Atos/Eviden and it is available on the IO-SEA GitHub. It has been deployed on the IO-SEA Pilot system cluster and integrated in the workflow manager API server for all ephemeral services.

A more high-level approach to orchestration can be also considered using a service such as Apache Airflow. A similar approach has been used in the LEXIS Platform, where Airflow is used to orchestrate workflows described by directed acyclic graphs in the form of DAG files [2, 3]. The DAG is composed of *Operators*, their parameters and relation (graph edges), which form the workflow tasks. Airflow then defines a *DAG Run* which is an instance of the workflow with a set of run-time parameters. Airflow provides a set of libraries of *Operators* in the form of *Providers*. The individual *Operators* are implemented as Python classes with a defined set of parameters passed at a task's execution time . The Airflow scheduler relies on an SQL database for orchestration of the task states, DAG runs

and other aspects. Airflow also provides its own UI and a REST API, which can be used for further integration in various platforms.

In IO-SEA this approach can be leveraged to create a *Provider*, which can implement various operations on the IO-SEA workflow manager and HSM stack as individual *Operators*. There are several possible levels of integration of the IO-SEA stack. Example workflow represented as Airflow DAG with tasks representing the operations in IO-SEA HSM and data move is presented in Figure 1.

The simplest one would be to implement the operations exposed by the WFM API and WP4 HSM as individual tasks in the Airflow DAG. This way a workflow with data movement operations can be defined, with the WDF file passed as parameter, parsed and corresponding data movement tasks triggered based on the user input. A proof-of-concept of this approach will be created and made available on the IO-SEA GitHub.

More enhanced level of integration would be the actual translation of a WDF file to the Airflow DAG, which would then trigger the individual steps of the workflow. Integration of this orchestration solution implies also proper handling of user authentication and access control, which may incur a lot of implementation overhead due to differences between the various APIs and access control systems used through the IO-SEA stack.

2.7. Error handling

The IO-SEA run time environment has been designed from the beginning to handle faults and failures as much as possible, even if we did not implement the most complex fault handling and recovery mechanisms. Sessions being a stateful concept, it is important for users to be able to recover from most of the possible hardware and software components failures.

Up to version 1.6, all software components run with user credentials on the login node with one exception: the resource manager deployed as a restful service on the dp-ioi service node (the service node hosting IO Instrumentation services). Note that nothing prevents running the API server and the database on a different node, providing it is able to submit jobs to Slurm.

Each user runs its own API server that implements transactional state changes stored in the per-user database. The database engine is a simple library and the database is stored as a file. Any failure, reboot, ... occurring on the login node between workflow manager commands has no effect on active sessions. When the API server is (re-)started, it reads the database and restores the current user sessions so that users can submit more steps and terminate them. The API server also (re-)creates the database file if it does not exist. It is a convenient way for users to resolve out-of-sync situations.

If a failure happens during the execution of a workflow manager command, it may lead to out-of-sync states. For instance, a step may have been submitted to Slurm but not recorded in the database. Gracefully managing such situations is not a significant issue; however, we have chosen not to address them in the 1.x versions of the Workflow Manager, as the architecture is slightly different starting from version 2.0.

Indeed, V2.0 relies on a centralized API server and orchestrator that will handle cli requests and submit Slurm commands on behalf of users, so it needs a more sophisticated user and credential management than V1.x, as well as a fault tolerance architecture. These aspects have been considered

in the project when selecting the base components, such as Apache Airflow for the orchestrator, that integrates fault tolerance features.

3. ParaStation Modulo Integration

The ParaStation Modulo software suite of ParTec has been enhanced to support health checking with the ParaStation HealthChecker for Ephemeral Services on datanodes of the IO-SEA Pilot system in combination with the IO-SEA WFM. In addition, launching jobs that use Ephemeral Services via Slurm SPANK plugins in ParaStation Management has been enabled. In the following sections, these enhancements are described in more detail.

3.1. ParaStation HealthChecker Integration

Health monitoring of the datanodes infrastructure is critical to keep the system up and running. An effort to integrate specific datanodes and ephemeral services metrics into the WP3 health monitoring framework has been done and is described in this section.

The ParaStation HealthChecker has been extended with a new API to check ephemeral services on datanodes. Details about this API are provided in Deliverable D3.3. The new ParaStation HealthChecker API has been integrated with the IO-SEA Workflow Manager to perform health checks on datanodes at reasonable points in time of a workflow. A new attribute `healthchecker` is defined in the `datanodes.conf` file on the Slurm controller node. This attribute defines the path of the ParaStation HealthChecker binary `pshcservcheck` to be run on the datanodes. ParaStation HealthChecker is started on datanodes at the start of a job (synchronous mode) and by a `cron` job (asynchronous mode).

On ParaStation HealthChecker side, efforts to provide health checks for both SBB and Ganesha ephemeral services have been made. The ParaStation HealthChecker extensions are designed in a modular way so that other ephemeral services could be added in the future. Due to time limitations of the project, the workflow manager focused on the integration and demonstration of the ParaStation HealthChecker in the workflow phases for Ganesha.

3.1.1. Synchronous Mode

In this mode, ParaStation HealthChecker checks that the resources are in the expected states for the current burst buffer. For each job requiring an SBB or Ganesha burst buffer (`create_persistent`, `use_persistent`, `destroy_persistent`), the burst buffer plugin starts the scripts `sbb.sh` or `gbf.sh`.

ParaStation HealthChecker is called at the end of each `stage-in` phase on a datanode when a persistent burst buffer (SBB or Ganesha) is *created*, *used* or *destroyed*. Just before the end of the `stage-in` phase, the `sbb.sh` or `gbf.sh` script remotely calls, through `ssh`, the `sbbdctl` or `gbf_server.sh` script on the datanodes using the `healthchecker` attribute of the `datanodes.conf`. Each time ParaStation HealthChecker is called once for a database operation (*create*, *use*, *destroy*) to keep track of the ephemeral services running on the datanode, and once for the actual `check` operation. The `check` command accepts two parameters: the path of the ParaStation HealthChecker binary and the identifier of the burst buffer. In synchronous mode, a message is written to a log file in

case of an error and the burst buffer ParaStation HealthChecker has been called for is moved to the *teardown* state, preventing it from being used by a further step run by the Workflow Manager.

3.1.2. Asynchronous Mode

In this mode, ParaStation HealthChecker checks that the ephemeral services are in the expected states for all the defined burst buffers. This is done with a `cron` job on the datanodes and the `check all` API call of the `pshcservcheck` script. The ephemeral services database of ParaStation HealthChecker is used to determine all ephemeral services that should be running on a datanode and then execute the defined checks for each registered service. For a list of all available checks see Deliverable D3.3. Similarly to the synchronous mode, any issues detected by the ParaStation HealthChecker are stored in the database and can be used from there for further processing by monitoring tools and system administrators.

3.1.3. Exemplary Content of Ephemeral Services Database

The following listing shows the exemplary content of the Ephemeral Service Database of ParaStation HealthChecker on one of the datanodes. It contains one Ganesha service named `toto`. The service information including the `sid` is provided upon creation of the service on the datanode while `state` shows that the service is currently *in use*. At the last check, the result of the check did not show any errors for the service.


```
[
  {
    "id": {
      "uid": 1028,
      "gid": 100,
      "mode": "gbf",
      "name": "toto"
    },
    "flash": [
      {
        "lv": "gbf_471967_10.2.19.211_stg",
        "vg": "MyContainer2",
        "mountpoint": "/run/gbf_ganesha.sh/13215/data",
        "size": 762
      }
    ],
    "daemons": [
      {
        "pid": 20668
      }
    ],
    "sid": "471967",
```

```
23     "state": "in use",
24     "check_date": "2024-02-13 10:51:13.135165",
25     "check_results": ["OK"]
26   }
27 ]
```

3.2. ParaStation Management Integration

The ephemeral services Smart Burst Buffer and Ganesha both consist of three parts. The first part is executed on the Slurm controller node, the second part is executed on the datanodes while the third part is loaded as a SLURM SPANK plugin on the compute nodes. The focus of this section is set on the third part. The ParaStation Management plugin called `psslurm` replaces all components of Slurm that are running on each compute node, namely `slurmd` and `slurmstepd`.

As discussed in Deliverable 2.2 [1] we decided to further refine the SPANK implementation of `psslurm` to enable the execution of the ephemeral services on the IO-SEA Pilot system. The first challenge was to allow the SPANK hooks `spank_init`, `spank_init_post_opt` and `spank_user_init` to modify the environment of corresponding user processes. As described in detail in Deliverable 2.1 [4], `psslurm` executes the different hooks in processes which cannot inherit from each other. Since this is also true for static variables it is especially important that there is an easy and reliable mechanism available to forward information between the hooks. The solution implemented transfers every change of the SPANK environment using psid messages.

For this approach the start of the child processes must be prevented so that the SPANK hooks can modify their environment. We achieved that by delaying the spawn messages until the SPANK hooks in the `psslurmforwarder` process finished. The `psslurmforwarder` process sends each modification of the environment back to the main `psslurm` daemon directly changing the step structure. When the spawn messages to start the children of the step are released, the environment is combined together. This makes it available in later SPANK calls executed in the `psidforwarder` (one dedicated forwarder for every child). To transfer the state of static variables between hooks, one can make use of this mechanism wrapping them in the SPANK environment.

Additionally, we allow the SPANK hooks to be executed not only in the context of a step but also in the context of a batch script and in prologue and epilogue. This includes calling the hooks at the right time of the initialization of those scripts as well as forwarding the required data structures which may be queried using the SPANK interface. The SBB and Ganesha SPANK plugins were executed successfully with these modifications on the IO-SEA Pilot system.

4. Lessons Learned

Different versions of the IO-SEA run time environment have been deployed on the IO-SEA Pilot system and exposed to WP1 use cases. For each version, most of them have been able to run and have provided valuable feedback described here.

The lessons learned during this integration effort are shared in the following sections.

4.1. DEEP test system integration

While being able to expose our tools to use cases on a near-production level system has been a tremendous source of feedback, it has generated a lot of overhead in terms of time to spend and latency to get components deployed and configured. Indeed, deploying system level tools requiring root access is not possible without involving system administrators, who are usually very busy. Debugging and fixing issues also usually require privileges that cannot be given to external users. Fortunately, we had root level access to the whole Sage prototype and it helped a lot with datanodes integration and deployment. The main difficulty has been the Slurm integration for which we had to go through the ticket system for software deployment and debugging. We want to thank here the DEEP system administrator team for their kindness and patience in these tasks.

To mitigate the effect, we decided early in the integration phase to limit as much as possible the use of root level requirement in our developments. For instance, we decided to cancel the central Workflow Manager API server that would require root privilege to submit jobs to Slurm on behalf of users, and instead developed a user level component that each user had to launch himself under its own credentials. It made the lives of our users a bit more complex (start/stop wfm_api, ...), but was manageable.

This will be taken into account in future projects (the same issue is happening in EUPEX) in which we will try as much as possible to develop and run our contributions in user mode.

4.2. Workflow Manager API behavior

The WFM API developed in the project has evolved several ways. These changes were motivated by the feedback we received from our users.

- Services synchronous start and stop: in the very first version of the Workflow Manager, starting or stopping an Ephemeral Service inside a starting or a stopping session command was done synchronously. This often caused timeouts when the test cluster was busy, the Workflow Manager not answering the CLI fast enough. This issue was solved by implementing asynchronous versions of the ephemeral services start and stop commands.

- Lack of dependency between the jobs that create or destroy the ephemeral services and the steps that use these services. This issue appeared on the test cluster when it was busy: in that situation, at session start, the service creation job was queued, waiting for resources, to be executed later. Then, when a step was run inside this session to start a job using the ephemeral service, that job might sometimes be scheduled by SLURM to start before the service creation job, leading to errors because the used service was not available yet. This issue was solved by adding dependencies between the various jobs inside a given session: now, the steps jobs, or even the service destroy job cannot start before the service creation job has successfully completed. Note that such dependencies are added to those that have potentially already been defined by the user.
- Sessions and steps without an ephemeral service: in the very first version of the Workflow Manager, it was forbidden to specify a step, or even a session, without an Ephemeral Service associated to it. We rapidly had to allow this feature in order for WP1 use cases to be able to establish a comparison between running their applications on top of an Ephemeral Service and running these same applications without relying on an ephemeral service.
- Heterogeneous jobs support: the TSMP use case needed to run heterogeneous SLURM jobs for the following reason: they needed to run several binaries at the same time, some of them requiring the same number of nodes and the same number of tasks per node, while the others needed a different number of nodes with one GPU on each. SLURM manages this kind of jobs in a specific way, particularly from the point of view of their *status*: the status of a heterogeneous job is not a single status string, but a string that concatenates the status of each of its components. We had to update the Workflow Manager API accordingly, combining the statuses of its individual components into a single one, in order for the TSMP use case to be able to run successfully.
- Status report format: in early versions of the Workflow Manager, the **iosea-wf status** command accepted many options and sub-options, with various combinations between them. WP1 use cases complained about this complexity that made this command difficult to use. We have, thus, added the possibility to run this command without any option, making it output a global detailed status of all started sessions, with their associated services and steps status.

4.3. Smart Burst Buffer ephemeral service

The Smart Burst Buffer Ephemeral Service is the simplest one to handle, because it does not impose datasets. It mostly accelerates data accesses in the sub-directory tree passed as the *targets* attribute at start time.

However, it is not always easy for use cases to anticipate this directory, as for instance, it may be dynamically created as part of the initial configuration step of the workflow. We had the case with the ECMWF use case, which caused the Ephemeral Service job creation to fail. It has no consequence because the Ephemeral Service runs correctly once the targets directory exists; however, it was confusing for users.

4.4. Slurm Burst Buffer framework integration

The workflow manager relies on the Slurm "Burst Buffers" plugin mechanism to create, manage and delete the ephemeral service. This service, being neither very mature nor widely used, presented some problems for users interacting with its interface. In particular, the `scontrol show bbstat` command reporting the global state of the datanode resources and the configured ephemeral services is very useful. It lets the users check the state of datanodes resources and understand why an Ephemeral Service cannot be launched if there are not enough resources left.

The listing below demonstrates a "normal" output of the command, when only a few services are launched.

```
[couvee1@deepv ~]$ scontrol show bbstat
FA: Total storage : 10TB
FA: Used storage : 0
FA: Free storage : 10TB
FA: Total memory : 400GB
FA: Used memory : 0
FA: Free memory : 400GB
[couvee1@deepv ~]$
```

However, over time the state of the Burst Buffer resources from the Slurm point of view became chaotic for various reasons:

- Errors happening within the Slurm plugin context leading to the TEARDOWN state. This state can only be cleaned up by a system administration operation.
- Errors happening on the Slurm client side, or in the Workflow Manager, leaving datanode resources not reclaimable by the user.
- User errors, destroying their own database with active sessions for instance
- Slurm errors, with for instance ephemeral services correctly terminated, but still being displayed in the staged-out state after weeks while they are supposed to remain displayed for only a few minutes

An example of such a chaotic display on the IO-SEA Pilot system is given here for illustration. We decided not to spend time on this during the project to dedicate our efforts to implementing new features. For a more accomplished solution, it would be mandatory to address this point.

```
[couvee1@deepv ~]$ scontrol show bbstat
FA: BB Type=GBF bbid=470226 Name=derr1-iosea-migration-0FbjCE7-gbf1 State=staged-in
    CreateTime=2024-01-31T11:00:22 Memory=0 StorageSize=1073741824 UserId=21944(derr1)
    Reason="Global Bunch of Flash ready"
FA: BB Type=GBF bbid=464151 Name=derr1-iosea-migration-OfYx7Xl-gbf1 State=staged-in
    CreateTime=2024-01-24T14:13:29 Memory=0 StorageSize=21474836480 UserId=21944(derr1)
    Reason="Global Bunch of Flash ready"
FA: BB Type=SBB bbid=457387 Name=couvee1-test1-sbb1 State=staging-out
    CreateTime=2023-12-15T14:27:17 Memory=10000000000 StorageSize=500000000000
    UserId=21898(couvee1) Reason="Waiting for SBB inactivity"
FA: BB Type=GBF bbid=463861 Name=derbey1-session009-testhestia009 State=staged-in
    CreateTime=2024-01-23T11:43:37 Memory=0 StorageSize=838860800 UserId=22841(derbey1)
    Reason="Global Bunch of Flash ready"
```

```
FA: BB Type=GBF bbid=455082 Name=wong6-ecmwf_nfs_v2-ecmwf-nfs State=staged-in
    CreateTime=2023-12-05T12:29:07 Memory=0 StorageSize=53687091200 UserId=22466(wong6)
    Reason="Global Bunch of Flash ready"
FA: BB Type=GBF bbid=470229 Name=derr1-iosea-migration-fZjN54O-gbf1 State=staged-in
    CreateTime=2024-01-31T11:20:43 Memory=0 StorageSize=1073741824 UserId=21944(derr1)
    Reason="Global Bunch of Flash ready"
FA: BB Type=GBF bbid=459364 Name=gregory1-nfs-bm-2024-01-08-143518-ID2-lqcd-nfs
    State=staged-in CreateTime=2024-01-08T14:36:19 Memory=0 StorageSize=53687091200
    UserId=4683(gregory1) Reason="Global Bunch of Flash ready"
FA: BB Type=GBF bbid=470217 Name=derr1-iosea-migration-ebwp1sb-gbf1 State=staged-in
    CreateTime=2024-01-31T10:29:24 Memory=0 StorageSize=1073741824 UserId=21944(derr1)
    Reason="Global Bunch of Flash ready"
FA: BB Type=GBF bbid=464127 Name=EphemeralBB State=teardown
    CreateTime=2024-01-24T12:19:05 Memory=0 StorageSize=21474836480 UserId=21944(derr1)
    Reason="Unable to wait for completion"
FA: BB Type=GBF bbid=474084 Name=furmanek1-dasi_ses-testdasi State=staged-in
    CreateTime=2024-02-13T11:04:43 Memory=0 StorageSize=6442450944
    UserId=21736(furmanek1) Reason="Global Bunch of Flash ready"
FA: BB Type=GBF bbid=456340 Name=wong6-ecmwf_nfs-ecmwf-nfs State=teardown
    CreateTime=2023-12-11T14:54:40 Memory=0 StorageSize=53687091200 UserId=22466(wong6)
    Reason="Failed to stop and release the filesystem"
FA: BB Type=GBF bbid=465665 Name=derr1-iosea-migration-J9UFtoh-gbf1 State=staged-out
    CreateTime=2024-01-25T11:32:22 UserId=21944(derr1) Reason="Resources released"
FA: BB Type=GBF bbid=459329 Name=furmanek1-CryoEM_60-cryoem-nfs1 State=teardown
    CreateTime=2024-01-08T13:29:05 Memory=0 StorageSize=0 UserId=21736(furmanek1)
    Reason="Failed to allocate resources - "
FA: BB Type=SBB bbid=462163 Name=caviedesvoullieme1-tsmp_msa_sbb-TSMP_SBB
    State=teardown CreateTime=2024-01-16T16:17:02 Memory=10000000000 StorageSize=0
    UserId=18194(caviedesvoullieme1) Reason="Stage out failed"
FA: BB Type=GBF bbid=453853 Name=wong6-ecmwf_nfs_v2-ecmwf-nfs State=staged-out
    CreateTime=2023-11-27T09:50:32 UserId=22466(wong6) Reason="Resources released"
FA: BB Type=SBB bbid=473130 Name=gregory1-sbb-bm-2024-02-09-160348-ID1-lqcd-sbb1
    State=staging-in CreateTime=2024-02-09T16:03:54 Memory=150000000000
    StorageSize=3000000000000 UserId=4683(gregory1) Reason="Waiting for SBB Resources"
FA: BB Type=GBF bbid=469950 Name=derr1-iosea-migration-ujdcZpr-gbf1 State=staged-in
    CreateTime=2024-01-30T11:46:59 Memory=0 StorageSize=1073741824 UserId=21944(derr1)
    Reason="Global Bunch of Flash ready"
FA: BB Type=GBF bbid=456692 Name=gregory1-nfs-bm-2023-12-13-122541-ID2-lqcd-nfs
    State=staged-in CreateTime=2023-12-13T12:26:25 Memory=0 StorageSize=53687091200
    UserId=4683(gregory1) Reason="Global Bunch of Flash ready"
FA: BB Type=GBF bbid=473837 Name=bozga1-dasi_ses-dasi2 State=staged-in
    CreateTime=2024-02-12T16:14:30 Memory=0 StorageSize=2147483648 UserId=20739(bozga1)
    Reason="Global Bunch of Flash ready"
FA: BB Type=GBF bbid=462668 Name=derbey1-testgbf009 State=staged-in
    CreateTime=2024-01-18T12:04:20 Memory=0 StorageSize=2147483648
    UserId=22841(derbey1) Reason="Global Bunch of Flash ready"
FA: BB Type=SBB bbid=462028 Name=tashakor2-testhi-TSMP_SBB State=staged-in
    CreateTime=2024-01-16T10:48:30 Memory=100000000000 StorageSize=2000000000000
    UserId=19755(tashakor2) Reason="SBB running on 2 servers"
FA: BB Type=GBF bbid=454377 Name=derr1-iosea-migration-eQpv8yB-gbf1 State=staged-out
    CreateTime=2023-11-29T12:34:34 UserId=21944(derr1) Reason="Resources released"
```

```
FA: BB Type=GBF bbid=456745 Name=gregory1-nfs-bm-2023-12-13-142933-ID2-lqcd-nfs
    State=staged-in CreateTime=2023-12-13T14:30:17 Memory=0 StorageSize=53687091200
    UserId=4683(gregory1) Reason="Global Bunch of Flash ready"
FA: BB Type=GBF bbid=460185 Name=gregory1-nfs-bm-2024-01-09-170100-ID2-lqcd-nfs
    State=staged-out CreateTime=2024-01-09T17:01:54 UserId=4683(gregory1)
    Reason="Resources released"
FA: BB Type=GBF bbid=459335 Name=furmanek1-CryoEM_61-cryoem-nfs1 State=teardown
    CreateTime=2024-01-08T13:41:30 Memory=0 StorageSize=0 UserId=21736(furmanek1)
    Reason="Failed to allocate resources - "
FA: BB Type=GBF bbid=471967 Name=toto State=staged-in CreateTime=2024-02-06T15:42:26
    Memory=0 StorageSize=799014912 UserId=22836(boudier1) Reason="Global Bunch of Flash
    ready"
FA: BB Type=GBF bbid=451731 Name=couvee1-ganesha4G-nfs1 State=staged-out
    CreateTime=2023-11-17T10:23:46 UserId=21898(couvee1) Reason="Resources released"
FA: BB Type=SBB bbid=456326 Name=gregory1-sbb-bm-2023-12-11-135824-ID1-lqcd-sbb1
    State=staged-out CreateTime=2023-12-11T13:58:33 UserId=4683(gregory1) Reason="Stage
    out completed"
FA: BB Type=SBB bbid=462663 Name=derbey1-session090-TSMP_SBB State=staged-in
    CreateTime=2024-01-18T11:24:17 Memory=10000000000 StorageSize=0
    UserId=22841(derbey1) Reason="SBB running on 2 servers"
FA: BB Type=SBB bbid=460182 Name=gregory1-sbb-bm-2024-01-09-170100-ID1-lqcd-sbb1
    State=staged-out CreateTime=2024-01-09T17:01:07 UserId=4683(gregory1) Reason="Stage
    out completed"
FA: BB Type=GBF bbid=473836 Name=bozga1-dasi_ses-dasi1 State=staged-in
    CreateTime=2024-02-12T16:14:30 Memory=0 StorageSize=2147483648 UserId=20739(bozga1)
    Reason="Global Bunch of Flash ready"
FA: BB Type=SBB bbid=461043 Name=derr1-simon3-TSMP_SBB State=staged-in
    CreateTime=2024-01-11T15:15:36 Memory=10000000000 StorageSize=0 UserId=21944(derr1)
    Reason="SBB running on 2 servers"
FA: Total storage : 10TB
FA: Used storage : 5693557494KiB
FA: Free storage : 4072067506KiB
FA: Total memory : 400GB
FA: Used memory : 290GB
FA: Free memory : 110GB
[couvee1@deepv ~]$
```

Part II.

Non-Volatile Memory Studies

5. Non-Volatile Memory usage in the IO-SEA stack

This second part describes the results of the studies conducted to explore the usage of Non-Volatile Memory technology in our IO-SEA architecture. Due to the major evolution of the situation in the industry (both Micron and Intel abandoned their products), we proposed an evolution of the task 2.2 to focus on the emerging CXL technology at month 18 and described the first results in deliverable 2.2.

In May 23, Seagate (in charge of this task) left the project very suddenly and no-one is available anymore to describe the additional studies they have done after deliverable 2.2. Eviden, as a contributor to this task, was focusing on the system integration on the datanodes. Only this aspect will be described in this section.

On Eviden side, we focused on the “Methods to build and deploy ephemeral services on datanode, where there is potential to also access data at byte level granularity” and the outcomes of this study are presented in the rest of this section.

5.1. Ephemeral Service leveraging Non-Volatile Memory

Non-Volatile Memory provides a means for storing data that remains retrievable after a power cycle, while also allowing for byte-level access. After a power cycle, the system restarts in an initial state, relaunching processes that will in turn need to re-attach the Non-Volatile Memory spaces they were using. To manage the ownership and security aspects, the first solution developed by the industry was to leverage the file systems’ semantics that specifically address these aspects: Non-Volatile Memory spaces are exposed as files and their access is handled by the file systems, exactly as for standard POSIX files. Byte-level access semantics are proposed through the pre-existing POSIX memory mapping mechanism (mmap() system call).

The ephemeral services implemented in the project could be configured to manage their persistent storage on such a Non-Volatile Memory resource:

- The Smart Burst Buffer manages the NVMe storage on datanodes by configuring an XFS file system on it and storing large IO requests payload as files and small IO requests payloads in a shared file to optimize performance.
- For the NFS/Ganesha ephemeral service, a local copy of the dataset is transferred from the long term storage to a volume on the datanode. This volume could easily be dynamically created on a Non-Volatile Memory device.

As illustrated in previous deliverables from the SAGE2 project, the performance advantage of this mode of operation over a standard file system on top of fast NVMe devices is not obvious, as modern file systems are able to cache most of the IO requests in memory.

In terms of persistence, ephemeral services are not concerned, as there is no need to retrieve data after a power cycle of the datanodes. Remember that the failure recovery model of the ephemeral

services is the same as the failure recovery of the compute nodes: a failure in the IO-SEA stack will be seen by the application as an IO error, leading in most cases to a crash of the application. In such cases, the application will simply be restarted after fixing any issue if needed. This model is based upon the fact that in terms of data storage, a typical HPC workflow can be seen as a "transaction" between an initial state (a set of input files) and a final state with a set of result files. If a workflow step crashes during the process and needs to be restarted, it will likely restart from its input files, with the noticeable exception of checkpointing mechanisms. For checkpoint files that must be accessible after a crash to restart a step, the IO-SEA solution proposes a user-settable "hint" to warn the Ephemeral Service that the file should be pushed to the long-term storage as soon as possible. Of course, this feature depends on an Ephemeral Service capable of handling files individually.

The version of the NFS Ephemeral Service storing files individually in the backend (see Section 2.5) would be a better candidate to leverage Non-Volatile Memory on a datanode. Indeed, it relies on a Key/Value store engine (Redis in our implementation) that usually needs to store small chunks of data (the keys) and update indices tables to speed up later accesses. Recent research demonstrated that a KVS module can make an efficient use of Non-Volatile Memory, such as [5], [6] or [7].

Many Open Source implementations of such a NVM-targeted KVS are available:

- pmemkv from Intel (`https://github.com/pmem/pmemkv`)
- uDepot from IBM (`https://github.com/IBM/uDepot`)
- kv-store from Infineon (`https://github.com/Infineon/kv-store`)

In its first version, the DASI Ephemeral Service has been implemented in such a way that the data is stored in POSIX files. However, the DASI library supports different backend technologies, and at the end relies on two backend services: an object store for the data and a Key/Value store for the indices. As such, it would be a good candidate for Non-Volatile Memory as well.

An extended version of the IO-SEA run time environment will be deployed on the EUPEX prototype. It will be ported on the new European RHEA processor and extended with a new ephemeral service: the Parallax Key/value store from Forth [8].

5.2. CXL and datanodes composability

The emerging CXL technology has the potential to enable a brand new approach to compose servers, as the components of a computer will become dynamically attachable to a system. This has been introduced in the deliverable D2.2 [9]. Due to the Seagate leave, no further experimentation has been done with simulation solutions such as QEMU. Instead, a theoretical study has been done to analyze the implications of such composable datanodes on the IO-SEA stack.

Datanodes resources are managed by the "Resource Manager" component. It is aware of the datanodes resources allocated to sessions and the resources that are still available. It allocates those resources upon requests from the session manager.

With the current implementation, resources on datanodes are of 3 types: memory, CPU cores and NVMe storage space. It may be complex to reach an optimal use of the datanodes, because not all ephemeral services have the same system resource usage. A Smart Burst Buffer Ephemeral Service

needs cores, memory and optionally NVMe space (allocated in fixed size chunks through flavors), while the NFS Ephemeral Service only needs NVMe disk space. As ephemeral services of any type can share datanodes, it is not obvious how to allocate all datanodes resources at one point in time (for instance, cores and memory may be available on a datanode but not enough NVME disk space to configure another NFS service).

CXL composability could help improving the overall usage of the datanodes resources with a dynamic resource allocation approach. Critical resources for datanodes would be pooled and dynamically assigned depending on the ephemeral services to configure.

5.3. perspectives

The Non-Volatile Memory ecosystem is still today more a research topic than a mature technology widely used in various domains. The software stacks to expose its unique features have mostly been derived from existing concepts (memory mapped files, file systems) to allow a quick integration in existing products. More specific libraries are also available, but their usage is quite complex.

The ephemeral nature of the IO-SEA architecture makes its usage not obvious and in most cases, a standard RAM space is enough for our Ephemeral Services. However, the larger capacity compared to standard DRAM may be an advantage to consider in the future.

Part III.

Summary

6. Summary

This final deliverable of the Work Package 2 describes the studies and the implementation of the IO-SEA run time environment.

The studies of the Non-Volatile Memory technology and its possible usages in the IO-SEA architecture, have been deeply affected by the Seagate withdrawal. The CXL emulation studies have been stopped and only the prospective study on the CXL usage in our architecture has been done.

However, the implementation of the IO-SEA stack in charge of providing ephemeral I/O services to workflows has been successfully done through a strong collaboration between partners in the work package, as well as with other work packages.

A command line interface has been developed, deployed on the IO-SEA Pilot system and exposed to IO-SEA use cases. It has been successfully implemented by the WP1 use cases.

The command line interface interacts with a session manager, a resource manager and an orchestrator to submit jobs to Slurm. Those components activate the workflow instrumentation tools from WP3 and create ephemeral services to expose dataset stored as objects in the WP4 long term storage solution. A datanode health checking mechanism has also been developed to make the solution manageable.

An Ephemeral Service (DASI) enables the management of the WP5 DASI databases as a dataset, unifying data management features for all types of user data.

The effective collaboration among the project partners enabled us to deploy an operational environment on an 80-nodes test system, which has provided extremely valuable feedback.

List of Acronyms and Abbreviations

A

API	An Application Programming Interfaces (API) allows software to communicate with other software which support the same API.

C

CLI	Command Line Interface.

D

DAG	Directed Acyclic Graph.
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.
ES	IO-SEA Ephemeral Service concept.

H

Hestia	Generic API to access a tiered storage environment, developed in the work package 4.
HSM	Hierarchical Storage Management.

I

IT4I	IT4Innovations National Supercomputing Centre at VSB Technical University of Ostrava, Czech Republic.

L

LEXIS	The LEXIS project : `https://lexis-project.eu/`.

N

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

ParaStation HealthChecker Healthchecking framework of the ParaStation Modulo HPC middleware suite made by ParTec.

ParTec ParTec is one of the leading SMEs in the HPC domain in Europe.

POSIX Portable Operating System Interface is a family of standards specified by the IEEE Computer Society for maintaining compatibility between operating systems.

R

RM IO-SEA Resource Manager component.

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.

SM ???.

SPANK Slurm Plug-in Architecture for Node and job (K)control.

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] M. Golasowski, D. Vasiliauskas, M. Rauh, N. Derbey, and P. Couvée. IO-SEA D2.2 Ephemeral data access environment: First version of the solution. Technical report, IO-Software for Exascale Architectures, Mar 2023.

[2] Martin Golasowski, Jan Martinovič, Jan Křenek, Kateřina Slaninová, Marc Levrier, Piyush Harsh, Marc DERQUENNES, Frederic Donnat, and Olivier Terzo. *Chapter 2 The LEXIS Platform for Distributed Workflow Execution and Data Management*. Taylor & Francis, 2022.

[3] Lexis documentation. `https://docs.lexis.tech/`. Accessed: February 19, 2024.

[4] A. Lopez, S. Valat, S. Narasimhamurthy, and M. Golasowski. IO-SEA D2.1 Ephemeral data access environment: Concepts and architecture. Technical report, IO-Software for Exascale Architectures, Jan 2022.

[5] Chen Ding, Jiguang Wan, and Rui Yan. Hybridkv: An efficient key-value store with hybridtree index structure based on non-volatile memory. *Journal of Physics: Conference Series*, 2025(1):012093, sep 2021.

[6] Kaixin Huang, Jie Zhou, Linpeng Huang, and Yanyan Shen. Nvht: An efficient key–value storage library for non-volatile memory. *Journal of Parallel and Distributed Computing*, 120:339–354, 2018.

[7] Ruicheng Liu, Peiquan Jin, Xiaoliang Wang, Zhou Zhang, Shouhong Wan, and Bei Hua. Nvlevel: A high performance key-value store for non-volatile memory. In *2019 IEEE 21st International Conference on High Performance Computing and Communications; IEEE 17th International Conference on Smart City; IEEE 5th International Conference on Data Science and Systems (HPCC/SmartCity/DSS)*, pages 1020–1027, 2019.

[8] Giorgos Xanthakis, Giorgos Saloustros, Nikos Batsaras, Anastasios Papagiannis, and Angelos Bilas. Parallax: Hybrid key-value placement in lsm-based key-value stores. In *Proceedings of the ACM Symposium on Cloud Computing*, SoCC '21, page 305–318, New York, NY, USA, 2021. Association for Computing Machinery.

[9] 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.