EuroHPC-01-2019

IO-SEA

IO – Software for Exascale Architectures

Grant Agreement Number: 955811

D1.3

Application I/O strategy

Final

Version:	1.0
Author(s):	E. B. Gregory(FZJ), P. Couvée(ATOS))
Contributor(s):	D. Caviedes Voullième (FZJ), D. Chapon (CEA), M. Golasowski (IT4I), M. Holicki (FZJ), O. Iffrig (ECMWF), J. Novacek (CEITEC), G. Tashakor (FZJ)
Date:	June 30, 2022

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:	D1.3
	Deliverable Nature:	Report
	Deliverable Level: PU *	**Contractual Date of Delivery:** 30 / June / 2022
		Actual Date of Delivery: 30 / June / 2022
	EC Project Officer:	Daniel Opalka

*– 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:** Application I/O strategy	
	ID: D1.3	
	Version: 1.0	**Status:** Final
	Available at: https://www.iosea-project.eu/	
	Software Tool: LaTeX	
	File(s): IO-SEA-D1.3-report.pdf	
Authorship	**Written by:**	E. B. Gregory(FZJ), P. Couvée(ATOS))
	Contributors:	D. Caviedes Voullième (FZJ), D. Chapon (CEA), M. Golasowski (IT4I), M. Holicki (FZJ), O. Iffrig (ECMWF), J. Novacek (CEITEC), G. Tashakor (FZJ)
	Reviewed by:	N. Derbey (ATOS) G. Umanesan (SEAGATE)
	Approved by:	Exec Board/WP7 Core Group

Document Status Sheet

Version	Date	Status	Comments
0.1	02.05.2022	Outline approved	complete
0.2	09.05.2022	Template use-case chapter and doc structure	complete
0.6	09.06.2022	Draft ready for internal review	complete
0.7	16.06.2022	1st internal review complete	complete
0.8	23.06.2022	post-1st-review edits complete	complete
0.8	27.06.2022	2nd internal-review complete	complete
0.9	27.06.2022	post-2st-review edits complete	complete
1.0	27.06.2022	final draft ready for EU submission	final

Document Keywords

Keywords:	IO-SEA, HPC, Exascale, Software

Contents

List of Figures

Listings

Executive Summary

This IO-SEA deliverable reports the preliminary plans to run the application workflow of each scientific use case within the IO-SEA environment. We detail the choices of ephemeral storage services and proposed syntax to access the services in sample scripts. We also include a description of the recent, particularly fruitful co-design feedback from the use cases. In addition, we document the updated ephemeral services interface and provide a detailed menu of the proposed storage services, including examples, all from the perspective of the application user.

1. Introduction

This third deliverable report from IO-SEA Work Package 1 describes the state of integration of the five IO-SEA scientific use-case applications into the IO-SEA ephemeral services environment, and describes the ephemeral storage services each one will use.

Originally this deliverable report was due at the end of April 2022, with the mandate to detail for each use-case application the choice of modules and APIs developed in Work Packages 2 – 5. This task would be relatively straightforward if the API design were mature and well-documented early in the year. However, we recognised early on that these services would be rapidly evolving up to the submission of deliverables D3.1 [1], D4.1 [2], and D5.1 [3] also at the end of April 2022, as well as the completion of D2.1 [4], submitted in February 2022.

Recognising the difficulty of building workflow implementation plans with a moving interface design target, we requested and received a three-month extension to the due date of this deliverable.

This extension has indeed allowed the design of the IO-SEA ecosystem to stabilize at a more mature state. Particularly, the design of ephemeral services access environment, the part of the IO-SEA solution most exposed to users, has been updated significantly since the submission of the Deliverable 2.1 report. We have been able to take these updates into account in the production of this report.

We organize this report into three distinct parts, providing an overview of both our co-design process, and the updated ephemeral services user interface that has resulted.

In Part I, we give an overview of some of the ways that the needs of the scientific use cases have influenced the IO-SEA project. Starting the process of porting use-case workflows to the ephemeral services environment has resulted in a rich period of cross–work-package collaboration and co-design results which are detailed in Chapter 2.

In Part II we give a thorough description of the updates to the ephemeral services and workflow management design, from the perspective of the HPC user. In some sense, this is a first draft of the IO-SEA user manual. Its inclusion is intended to both as an aid to the reader in understanding the use-case application interfaces choices, and as a reference manual for the scientific use cases in planning their workflow designs.

Within this Part II, Chapter 3 describes the updated ephemeral services design interface, including the current standard for the syntax of YAML Workflow Description Files (WDFs) and the IO-SEA commands for creating data sets, namespaces and managing workflow sessions. Chapter 4 provides detailed descriptions of the six ephemeral storage services exposed to users, including relative criteria for choosing between them. In Chapter 5 we give detailed generic examples for how each of the services might be invoked in a workflow, with snippets from WFDsand associated command scripts to invoke them.

Finally, Part III contains a chapter for each of the five scientific use cases, which not only describe their choices of ephemeral services, but gives a first draft of the WDFs and command sequences used to run their workflows. This includes a plan for how the workflows I/O data is organized into data sets and namespaces. We do not consider this to be the final word on these plans, as the interface and service designs, as well as how they are employed by the use cases, will evolve in the coming

months. This is particularly expected as we move to deploy the IO-SEA environment and use cases to a prototype machine. We expect that the plans and documentation in the following pages should make this porting task much more straightforward.

Part I.

Co-design in IO-SEA

2. Co-design process and results

Since the start of the IO-SEA project in April 2021, there have been on-going discussions between the developers of the IO-SEA technical solution in Work Packages 2 – 5 and the collaborators in Work Package 1, who concern themselves with specific scientific use-case applications. Generalizing, the former are primarily computer scientists and the latter computational scientists in a specialized field.

There is a large area of knowledge overlap. However, the differing areas of expertise mean that communication and education must flow both ways for these groups to successfully complement each other to collaborate in developing and testing the IO-SEA storage environment.

2.1. Initial co-design discussions

In the first months of the project these discussions included, a full day cross-work-package meeting and discussion, as well as presentations and discussions with the leaders of the technical work packages in the regular Work Package 1 meetings Additionally, there were many bilateral meetings between members of a specific use-case task and the collaborators of one of the other technical work packages.

The topics of these discussions and presentations in the early months centred on the general I/O needs of a use-case application, what the technical work packages planned to develop, and how the latter might be matched to the former.

The first stages of this co-design process are described in IO-SEA deliverable D1.1: Application and co-design input [5].

As the proposed design of the IO-SEA technical solution has matured, we have begun to work out the details of how the scientific use-case applications would be run within the IO-SEA environment.

Beginning with lattice quantum-chromodynamics (LQCD) as a test case, deep-dive discussions between Work Package 2 and the use-case tasks allowed the use cases to map out how they might bundle their I/O files into data sets and namespaces, and how the workflow might be integrated into the IO-SEA workflow management system with YAML-format Workflow Description Files (WDFs).

These discussions took place both in cross–work-package workshops and in the bi-weekly Work Package 1meetings. Each of the five scientific use-case applications was discussed in detail, highlighting the unique features and data-access patterns of each. As the use cases began the trial exercise of porting their applications to the proposed IO-SEA ecosystem, it became clear that the workflow manager and IO-SEA ephemeral services must accommodate the variety of data access patterns needed for different workflows.

Some of these co-design discussions even involved the scientific use cases of DEEP-SEA [6] Work Package 1, who joined, for example, a discussion of DASI [3] from the users' perspective.

In a two-hour whiteboard session in the April 2022 IO-SEA All-Hands meeting we considered four phases of a computational project: "setup", "during the workflow run", "after the workflow run", and "other". The discussion revealed that in all four phases there is a need for various types of interactive

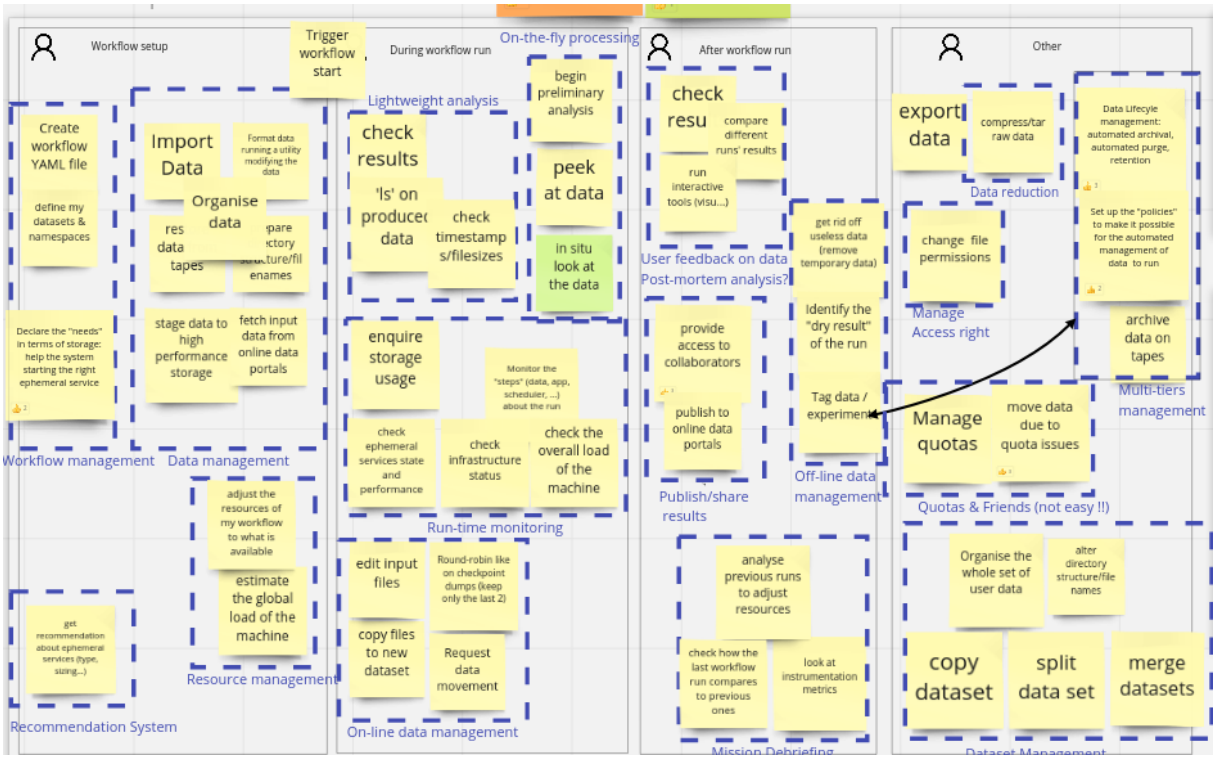

Figure 1: Screen capture of IO-SEA whiteboard co-design brainstorming session.

access to data sets and data movement that was not necessarily accommodated by the initial vision of the ephemeral services. Figure 1 shows a screenshot of the resulting whiteboard.

2.2. Co-design results to date

From the varied discussions between technical work packages and scientific use cases described above we can identify several features whose inclusion in the IO-SEA design plans have been influenced by the expression of the needs of the use cases.

These include:

- **Interactive data access**
  In all phases of the project, it is important for scientific use cases to have interactive access to the data sets and namespaces used in the workflow, e.g., to stage initial data in setup; to check output, peek at meta-data or even begin analysis while the workflow is running; to run interactive data analysis tools post-workflow. Work Package 2 has responded with a proposal for an `iosea-wf access` command, providing access to active data sets and namespaces though an interactive session. An example can be seen in the WDF in Listing 3.2 and also in the LQCD example in Listing 10.3
- **Prompted tape archival/retrieval**
  The experience of scientific users at large HPC centres is that the time to move large data

sets to or from tape can sometimes be measured in days. Therefore it is important that the users have an interface to trigger data retrieval from the tape long before a workflow is queued. The proposed solution is the *datamover*, illustrated in the `datamover2` service in Listing 3.7 in Chapter 3.

- **Tape-disk explicit copy**
  Much of the discussion regarding the IO-SEA storage hierarchy has used the phrase "data-movement". As the tape archive is a tier in the storage hierarchy — the slowest but most capacious tier, use cases clarified that they rarely "move" data to or from the tape archive systems. This occurs normally only in response to suddenly needing to make space on a nearly-full disk. Rather the normal pattern is often that data is generated that has both near-term and long term uses and a *copy* is saved to tape for safe-keeping, while the bulk may remain on disk for further immediate access. Likewise, when older data is retrieved from tape for new analysis, it is usually *copied* to disk; the tape archive is not erased. This clarification has resulted in the inclusion of copy functionality of the datamover, also illustrated in Listing 3.7
- **Pre-fetching to data node**
  Similarly, users in environments where compute-node time is charged against a budget expressed concern that they be able to pre-fetch files to the data nodes so that the compute nodes do not idle when the allocation starts. Again, the solution provided in response was the datamovers, which will also allow users to prompt data movement to the flash storage. The service `datamover1` is an example of this in Listing 3.7.
- **DASI-to-POSIX staging application**
  Several scientific use-case tasks expressed interest in exploring the proposed DASI as a method of organizing stored scientific data. However, to integrate DASI directly into the application generally requires significant re-writes to I/O libraries. In cases where the application is a large, widely-used community code, this is impractical and beyond the scope of the IO-SEA project.

  Instead, the use-case tasks have requested a stand-alone application that can translate a POSIX path to DASI semantic keys and copy a file back and forth between DASI and POSIX storage. This will enable us to explore the advantages of DASI with a low barrier to entry. Work Package 5 has indicated they intend to produce such an application in the coming months in response to this request. In anticipation of this, Section 5.1 provides an example of how this might be integrated into a workflow with the GBF ephemeral service.

2.3. On-going codesign process

With the first draft of the ephemeral services interface commands in Deliverable 2.1 [4] in February 2022, and the updates described in Chapter 3 of this report, the use cases are able to cast more realistic and complex models of their workflow into the IO-SEA ephemeral services syntax. Preparation for this report has prompted very energetic efforts to do so. The use-case tasks are now considering not just the normal workflow when everything runs smoothly, but the difficult and exceptional cases that we encounter as part of managing large computational projects.

This process has already prompted many questions regarding the feasibility of ever more complex data access patterns. When the response from Work Package 2 has not been immediately affirming

that the access is possible, they have either provided us with an equivalent, alternate access method, or promised to consider whether the interface design could be made to accommodate that pattern.

The interesting queries in this third, interesting category include, for example how active data sets might be shared between separate users (collaborators in a project, for example), and whether it might be able to add more data sets and steps to a WDF on the fly.

Also, upon seeing the new features, there has been a natural tendency to attempt to use them creatively, sometimes to the point of pushing past the design boundaries. This feeds back into the design loop and will sometimes generate improved features.

On a final note, during the process of integrating the use-case applications into the IO-SEA syntax many misconceptions have been exposed and errors made. This is a healthy reminder that there is a value to simplicity in interface design. The use-case personnel are all experienced HPC users, and our errors often indicate places where the interface syntax may also challenge less-experienced users in the future when the IO-SEA environment is introduced to wider use.

We expect this co-design process to continue, and even accelerate, as we begin to port applications into the IO-SEA environment on prototype machines. In the process of preparing this report the interface syntax has evolved slightly in response to the above described factors. We are now entering perhaps the most fruitful period of co-design in the IO-SEA project.

Part II.

Ephemeral services and workflow management

3. Ephemeral Services Update

Since the finalisation of Deliverable 2.1 [4], the proposed IO-SEA Workflow handling commands have evolved. We present them here in updated form to aid the reader in understanding the next chapters, where use cases describe their integration with the IO-SEA stack.

3.1. User commands

In order to implement the IO-SEA ephemeral services logic, workflows will be executed within *sessions*. A session needs to be started to allow launching the different steps of the workflow, and stopped when done through the commands presented in Listing 3.1.

```
# start a session for my workflow (described in the WDF.yaml file)
iosea-wf start WORKFLOW=WDF.yaml SESSION=My_Session

#run the workflow steps
iosea-wf run SESSION=My_Session STEP=step1
iosea-wf run SESSION=My_Session STEP=step2
iosea-wf run SESSION=My_Session STEP=step3

#stop the session and release the datanode
iosea-wf stop SESSION=My_Session
```

Listing 3.1: Workflow Session main commands

To start a session, a YAML file containing the description of the workflow (described in Section 3.2) and a user provided session name are needed.

The session name is used in the run commands to execute a step within a started session, allowing to have multiple active sessions in parallel if needed.

A few other commands are provided to manage sessions, presented in Listing 3.2. The `list` command reports all active session names. The `status` command reports the state of the active session passed as a parameter, listing all terminated and running jobs and ephemeral services. The `access` command sets up an interactive environment (a shell) in which all the ephemeral services of an active session are configured, giving access to the namespaces in interactive mode.

```
# list all active sessions of the user (report the session-names)
iosea-wf list

#display info about jobs & ephemeral services
iosea-wf status SESSION=My_Session

# Start an interactive access environment for all or limited to [<service>]
iosea-wf access SESSION=My_session
```

Listing 3.2: Workflow Session management commands

3.2. Workflow Description File

Users are requested to describe their workflow steps, and for each step the ephemeral services they need in a *Workflow Description File* (WDF) in YAML format with the following sections :

```yaml
workflow:
  name: My_Workflow

services:
  - name: ephemeral_service_1
    type: NFS
    attributes:
      namespace: My_namespace
      mountpoint: /mnt/USER/My_Workflow
      flavor: medium

steps:
  - name: step_A
    location:
      - gpu_module
    command: "srun My_Step_A"
    services:
      - name: ephemeral_service_1
```

Listing 3.3: Sample Workflow Description File in yaml

The name assigned to the workflow will be used as an attribute in the Work Package 3 instrumentation tools in order to facilitate retrieving metrics for the different sessions. Workflows will be presented in a table in which two columns (that can be filtered and sorted) will contain:

- the workflow name taken from the WDF.yaml file
- the workflow run ID built from the session name and a timestamp to differentiate between sessions having the same session names

Figure 2 illustrates such a workflow table.

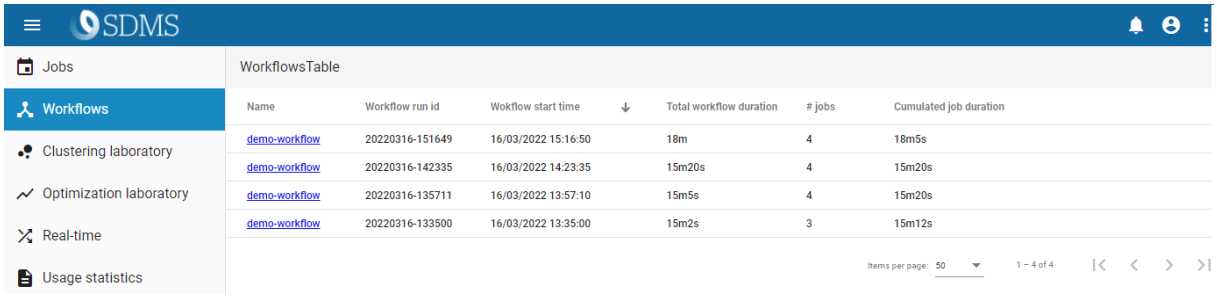

Figure 2: WP3 IO Instrumentation workflow table example

The `services` section of the WDF describes the ephemeral services needed to run this workflow. It contains the type of ephemeral service (detailed in Chapter 5), and for each type, the required attributes, such as the namespace to be made accessible and the mountpoint on which it will be

mounted on compute nodes for POSIX ephemeral services. An additional "flavor" attribute is proposed to indicate the data node resources to allocate for the service (CPU cores, memory, etc.).

The `steps` section of the WDF describes the different steps of the workflow. The main attributes are:

- `name`: Name used in the run command
- `location`: Indicates the target compute module(s) in the MSA architecture
- `command`: Indicates the command to be launched by the Workflow Manager on behalf on the user. It must be a command to be submitted to the Resource Manager of the target module. In IO-SEA, all prototypes will rely on Slurm, so it must be either a `srun` command, or an `sbatch` script (`salloc` is supported by the Workflow Manager through the access command presented earlier in this document)
- services: List of the ephemeral services to be configured on the allocated compute nodes

In the example in Listing 3.3, the namespace targeted by the `ephemeral_service_1` (`My_namespace`) is hard-coded in the WDF. This will prevent starting more than one session of this workflow at the same time, as the workflow manager will lock at session start time the data sets accessed in the session (to prevent multiple workflows writing to the same data set). To handle this case, it is possible to build a WDF that contains a variable instead of a hard-coded namespace, as illustrated in Listing 3.4.

```
workflow:
  name: My_Workflow

services:
  - name: ephemeral_service_1
    type: NFS
    attributes:
      namespace: "{{ NS1 }}"
      mountpoint: "/mnt/USER/My_Workflow/{{ SESSION }}/ES1"
      flavor: medium
  - name: ephemeral_service_2
    type: NFS
    attributes:
      namespace: "{{ NS2 }}"
      mountpoint: "/mnt/USER/My_Workflow/{{ SESSION }}/ES2"
      flavor: medium

steps:
  - name: step_A
    location:
      - gpu_module
    command: "srun My_Step_A"
    services:
      - name: ephemeral_service_1
  - name: step_B
    location:
      - gpu_module
    command: "srun My_Step_B"
    services:
      - name: ephemeral_service_2
```

Listing 3.4: Sample Workflow Description File with parametric namespace

Such a parametric WDF is handled by adding the actual namespaces for the session either at session start time, or even at step run time, as presented in Listing 3.5

```
# start a session for my workflow (described in the WDF.yaml file)
iosea-wf start WORKFLOW=WDF.yaml SESSION=My_Session NS1=My_namespace

#run a workflow step
iosea-wf run SESSION=My_Session STEP=step_A

# start an interactive access envionment
iosea-wf access SESSION=My_Session NS2=My_other_namespace

#run more workflow steps
iosea-wf run SESSION=My_Session STEP=step_B NS2=My_third_namespace
iosea-wf run SESSION=My_Session STEP=step_B NS2=My_fourth_namespace

#stop the session and release the datanode
iosea-wf stop SESSION=My_Session
```

Listing 3.5: Sample session handling with parametric WDF

The `access` command to launch an interactive session will configure only fully defined ephemeral services (i.e. having their namespace known by the Workflow Manager). If some parametric namespaces have not been defined yet at the time the access command is launched, they can be added as options on the command line (`NSx=...`).

```
steps:
  - name: step_A
    location:
      - gpu_module
    command: "srun -n {{ CORES }} My_Step_A"
    services:
      - name: ephemeral_service_1

#run the workflow steps
iosea-wf run SESSION=My_Session STEP=step_A CORES=10
```

Listing 3.6: Defining and using variables in the Slurm command line

To handle cases where users need to adapt the run time parameters of their steps dynamically, variables can also be defined in the WDF for the "`command`" attribute and instanciated at run time, for instance to adjust the number of cores to run a step, as illustrated in Listing 3.6.

3.3. Advanced features

Datamovers are proposed through the WDF to trigger data movements within a namespace between the layers of the hierarchical storage architecture before and after running steps (set in the trigger attribute). Before running steps, the goal is to ensure that the data to be accessed by these steps is actually located on the expected (fast) tier, or to flush data to lower (i.e. slower) levels to free space on fastest tiers. After running steps, this is useful to move or replicate critical data to lower performance (but secured) tiers. Datamovers are associated with ephemeral services (services field in the WDF) and the elements attributes must start from the root of the namespace. The full path for POSIX ephemeral services is obtained by concatenating the mount point of the ephemeral service and the elements attributes. Listing 3.7 gives an example of two datamovers defined for a NFS ephemeral service. The first, `datamover1`, is intended to ensure that the `/mnt/USER/My_Workflow/gauges/*.hdf5` and `/mnt/USER/My_Workflow/input/*` files will be located in the "flash" tier before starting the step, while `datamover2` is triggered when the step ends to move the `/mnt/USER/My_Workflow/gauges/*.hdf5` result files to a safe layer of the storage hierarchy (tapes here). Two operations are proposed:

- `move` : change the location of a data set in the storage hierarchy.
- `copy` : create a new copy of the data set in the specified tier while keeping the original data in the current tier.

```
services:
  - name: My_ephemeral_service
    type: NFS
    attributes:
      namespace: My_namespace
      mountpoint: /mnt/USER/My_Workflow
      flavor: medium
    datamovers:
      - name: datamover1
        trigger: step_start
        target: flash
        operation: copy
        elements:
          - "gauges/*.hdf5"
          - "input/*"
      - name: datamover2
        trigger: step_stop
        target: tape
        operation: move
        elements:
           "gauges/*.hdf5"


steps:
  - name: step_A
    location:
      - cpu_module
    command: "sbatch "
    services:
      - name: My_ephemeral_service
        datamovers:
          - datamover1
```

```
        - datamover2
```

Listing 3.7: Datamover service sample code

User provided *hints* are another advanced feature to optimise the performance of the I/O run time environment. They do not perform any actions but rather give information about the behavior of the workflow. This is still being defined but an example is given in Listing 3.8.

It can be useful to know for each step the access mode of ephemeral services in terms of read and write operations. The keywords `RO`, `RW`, `WO`, for 'read-only', 'read-write' and 'write-only', respectively, can be specified next to the namespace to give the information. For instance, a step accessing a service in `WO` mode doesn't need any data to be prefetched.

```
steps:
    -name: step_A
     location: gpu_module
     command: "sbatch My_App.sh"
     services:
       - name: My_Ephemeral_Service_1 RW
       - name: My_Ephemeral_Service_2 WO
```

Listing 3.8: User provided hints example

4. Menu of Ephemeral Services

There are two classes of ephemeral services exposed to Workflows:

The first are the storage services with no connection to the long term storage. These services start with empty POSIX file systems and all data they contain is destroyed when they are stopped. So, they are not associated with datasets.

- **SBF** (Single Bunch of Flash) for single compute node temporary POSIX file system.
- **GBF** (Global Bunch of Flash) for multiple compute nodes shared temporary POSIX file system.

The SBF ephemeral service creates a volume with an XFS file system on one data node. The XFS file system is later mounted on one single compute node at a time. An SBF ephemeral service can be used successively by many steps, but never at the same time.

The GBF ephemeral service allocates a volume on each allocated data node and configures a BeeGFS parallel file system on them. The GBF service can be used by steps running on many compute nodes, either successively or in parallel.

The attributes needed for SBF and GBF ephemeral services in the WDF are:

- `mountpoint`: the directory on the compute nodes where the ephemeral file system will be mounted
- `flavor`: defines the resources to allocate on data nodes for the service, in terms of CPU cores, RAM and fast storage (NVMe, NVRAM). This attribute is not fully defined yet and will be arbitrarily set to medium in the document
- `size`: size of the storage space
- `datanodes`: number of data nodes to use (only required for GBF)

The typical use of these ephemeral services is to store data that needs to be kept only for the duration of the session, such as progress logs or traces, convergence analysis. It is also possible to define a specific step in the workflow that will move data from this temporary storage space to a longer-term data set if needed. For instance, the step could copy the logs and traces for further analysis when the workflow is not behaving as expected by the user.

The second class of storage services includes those that provide access to data sets stored in the long term storage:

IO-SEA proposes three data set technologies:

- Data sets containing POSIX file systems
- Data sets containing S3 objects and buckets
- Data sets containing DASI scientific data

POSIX data sets are handled either by **NFS** (for standard performance POSIX file access) or by **BB-NFS** (Burst-buffered NFS, for high bandwidth/low latency POSIX file access). The NFS ephemeral service runs on one data node, while the BB-NFS ephemeral service adds a burst-buffer layer in front of the NFS server that can be deployed on many data nodes, allowing to significantly increase the bandwidth and provide lower latencies. IO-SEA users are recommended to experiment first with the standard NFS ephemeral service, and move to BB-NFS for the most demanding data sets after analysing the workflow behavior with the tools provided by Work Package 3.

POSIX data sets need the following attributes in the WDF:

- `namespace`: the namespace to be used to expose the data set
- `mountpoint`: the directory on the compute nodes where the namespace will be mounted
- `flavor`: defines the resources to allocate on data nodes for the service, in terms of CPU cores, RAM and fast storage (NVME, NVRAM). This attribute is not fully defined yet and will be arbitrarily set to medium in the document
- `size`: size of the storage space on data nodes
- `datanodes`: number of datanodes to use for the Burst Buffer layer (only required for BB-NFS)

When using POSIX data sets, users have nothing to configure on data nodes. The namespaces will be mounted by the run time environment on each compute node at the specified mountpoint.

S3 data sets are exposed by the CORTX S3 server, running on one data node. The following attributes are required in the WDF:

- `namespace`: the namespace to be used to expose the data set
- `server`: the environment variable that will be initialized with the address of the S3 server by the workflow run time environment
- `port`: the environment variable that will be initialized with the port to contact the S3 server by the workflow run time environment
- `flavor`: the amount of resources allocated for the S3 server on the data node (this attribute is not fully defined yet, use medium for now)

For S3 ephemeral services, users will need the IP address and the port to reach the S3 server just configured. The information will be passed back to the user in the provided environment variables that will be filled by the run time environment on each compute node. This is convenient when several S3 ephemeral services are needed for a workflow.

DASI does not really expose data sets, as it hides the low level storage details from end users. The initial version of DASI relies on a configuration file containing the information needed by the library to access scientific data. The name and location of this file will be the attribute for DASI ephemeral services:

- `configfile`: the name and path of the DASI configuration file
- `flavor`: the amount of resources allocated for the DASI gateway on the data node (to be defined later, use medium for now)

5. Ephemeral Services Examples

In this chapter, a few examples are presented, and more details are given on command behaviour and the conduct of a session.

5.1. Example with GBF and DASI ephemeral services

Listing 5.1 illustrates a workflow using a GBF service to store the files produced by the main workflow steps, and implementing a dedicated final step to save the important results into a data set. The WDF contains two ephemeral services (GBF and DASI), two steps for computations, and one more to move final data to a DASI database. A new "TAG" option is introduced in this listing and presented later in this chapter.

```
workflow:
  name: My_Workflow

services:
  - name: RunTimeData
    type: GBF
    attributes:
      mountpoint: /mnt/USER/My_Workflow
      size: 10TB
      flavor: medium
      datanodes: 4
  - name: ResultsDASI
    type: DASI
    attributes:
      configfile: /home/user1/.dasi/dasiConfig.txt
      flavor: medium

steps:
  - name: step_A
    location: gpu_module
    command: "srun My_Step_A"
    services:
      - name: RunTimeData
  - name: step_B
    location: cpu_module
    command: "srun My_Step_B"
    services:
      - name: RunTimeData
  - name: SaveResults
    location: cpu_module
    command: "srun My_Copy_Script"
    services:
      - name: RunTimeData
      - name: ResultsDASI


# start workflow
```

```
iosea-wf start WORKFLOW=WDF.yaml SESSION=My_Session

#run the workflow steps as needed
iosea-wf run SESSION=My_Session STEP=step_A TAG=init
iosea-wf run SESSION=My_Session STEP=step_B
iosea-wf run SESSION=My_Session STEP=step_B TAG=last

#run the final step to push the results in DASI
iosea-wf run SESSION=My_Session STEP=SaveResults

#stop the session and release the datanode
iosea-wf stop SESSION=My_Session
```

Listing 5.1: Example wih GBF and DASI

The `run` and `stop` commands are asynchronous, meaning that they return when the required action is submitted to the Workflow Manager, but have not necessarily executed yet. The `status` command will be used to follow the progress of those requests. A sample output of the `status` command is presented in Listing 5.2. The `status` command will report all steps in three categories:

- **Pending Steps** for jobs submitted to the IO-SEA Workflow Manager, but not scheduled yet by Slurm. Some may have already a Slurm Job ID (`step_B`), and some may even be still waiting to be submitted to Slurm, waiting for data node resources for instance (`SaveResults`).
- **Active Steps** for jobs scheduled by Slurm and currently running.
- **Terminated Steps** for terminated jobs

```
> iosea-wf status My_session
Workflow Name  : My_Workflow
Workflow SessionID : My_session_<timestamp>

Pending Steps
Step Name   Slurm ID    Tag     Command                 Reason
step_B      2767        last    srun My_Step_B          Waiting in Slurm Queues
SaveResults ----                srun My_Copy_Script     waiting Ephemeral Service

Active Steps
Step Name   Slurm ID    Tag     Command
step_B      2763                srun My_Step_B

Terminated Steps
Step Name   Slurm ID    Tag     Command                 Reason
step_A      2760        init    srun My_Step_A          Terminated OK

Ephemeral Services
....
```

Listing 5.2: Output of the status command

As shown here, the same step can be launched several times, sequentially or in parallel. For convenience, a `TAG` option is proposed to identify specific runs such as for instance the last run of a step.

The `Command` field will display the command as submitted to Slurm, allowing the user to check how variables have been processed by the Workflow Manager.

A parsable version of this output will be aloso available to ease integration in automation scripts.

5.2. Example with a multi-module application and on-the-fly processing

Listing 5.3 illustrates a workflow with a simple simulation step running on a compute module, a more complex post-processing step running on two compute modules (`cpu_module` and `gpu_module`) and a third data conversion step running on data nodes (on-the-fly processing).

```
workflow:
  name: My_Workflow

services:
  - name: TempData
    type: GBF
    attributes:
      mountpoint: /mnt/USER/My_Workflow
      size: 10TB
      flavor: medium
      datanodes: 4

  - name: Results
    type: BB-NFS
    attributes:
      namespace: Results
      mountpoint: /mnt/USER/My_Results
      Size: 10TB
      flavor: medium
      datanodes: 4

steps:
  - name: simulation
    location:
      - cpu_module
    command: "sbatch My_Simulation"
    services:
      - name: TempData

  - name: postprocess
    location:
      - cpu_module
      - gpu_module
    command: "sbatch -d after:{{ SIMid }} My_Post_Processing"
    services:
      - name: TempData
      - name: Results

  - name: dataconvert
    location:
      - datanodes
    command: "sbatch -d after:{{ SIMid }} My_Data_Conversion"
```

```
    services:
      - name: Results

# start workflow
iosea-wf start WORKFLOW=WDF.yaml SESSION=My_Session

#run the workflow simulation step
iosea-wf run SESSION=My_Session STEP=simulation

# Wait until simulation job is submitted to Slurm to get its Slurm Job ID
iosea-wf status SESSION=My_Session

#run the postprocessing and data conversion steps in parallel when simulation is done
iosea-wf run SESSION=My_Session STEP=postprocess SIMid=<simulation step job ID>
iosea-wf run SESSION=My_Session STEP=dataconvert SIMid=<simulation step job ID>

#stop the session and release the datanode
iosea-wf stop SESSION=My_Session
```

Listing 5.3: Multi-Module Application

The goal of the "on-the-fly processing" mode is to reduce data movement between compute nodes and data nodes. For instance, consider a multi-node application creating one huge, shared data file with write operations from all processes, and a post-processing application that will create a JPEG image from this huge data file. Running the post-processing application on another compute node would require moving the huge data file to its memory. Running the post-processing application on the data node that manages it will eliminate this data movement, improve time to solution, and lower energy consumption. To enable launching steps on data nodes, it will be necessary to create a dedicated partition in Slurm and dedicate a part of the data nodes compute resources (CPU cores, RAM) to jobs. Accelerators such as GPUs and FPGAs could be used as well for on-the-fly processing steps.

Locating a step on data nodes is managed by setting the `location` field to “`datanodes`”. However, such a step can only use one ephemeral service, in order to be able to identify the target data nodes.

Part III.

Use-case ephemeral service choices

6. Astrophysics with RAMSES

6.1. Use-case overview

In this chapter we describe the IO-SEA ephemeral services likely to be used during a RAMSES simulation workflow for a typical astrophysical use case (cosmological simulation, isolated galaxy model, interstellar medium evolution, supernova explosion, etc). The typical workflow in RAMSES astrophysical simulations has already been described in previous deliverable reports [5, 7] and is illustrated in the workflow diagram in Figure 3.

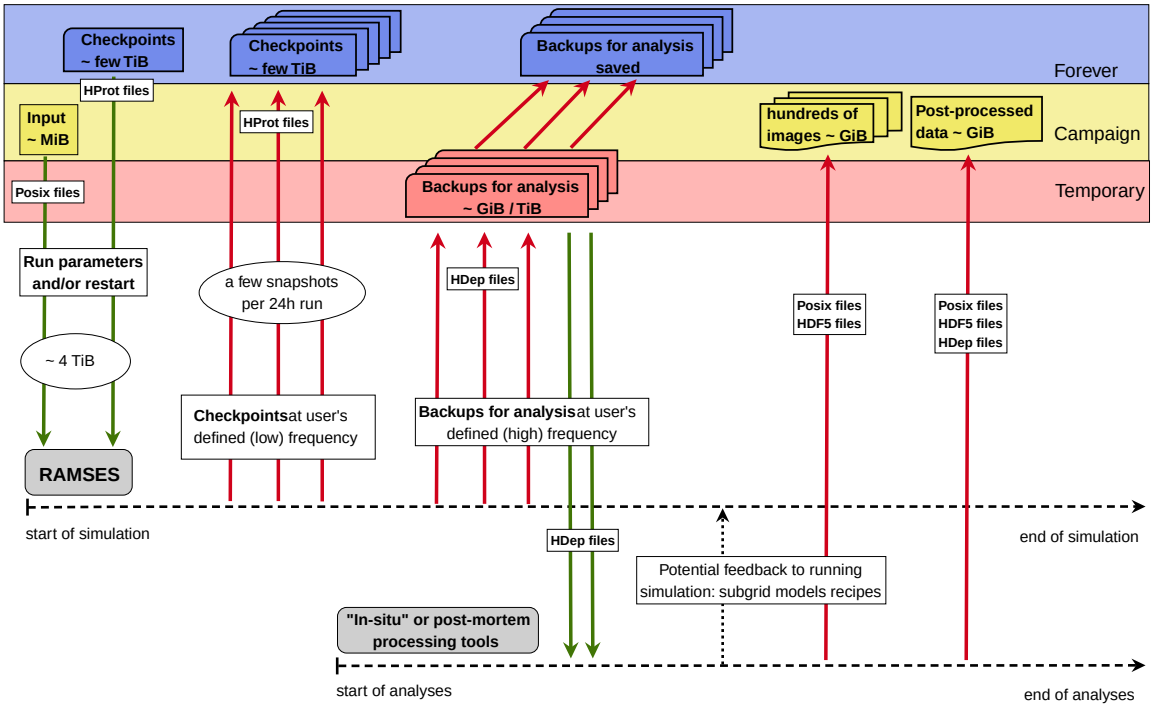

Figure 3: RAMSES astrophysical simulation workflow diagram, showing data movement through the various workflow steps and storage hierarchy.

After initialisation of the 3D model from initial condition files and simulation configuration files, RAMSES has two main I/O data flows: checkpoints (used upon restart) and backups (for post-processing purposes only). Both are handled by the Hercule parallel I/O library. Because of the use of Adaptive Mesh Refinement (AMR) in RAMSES, the amount of data to transfer on each workflow is not static and will evolve with the dynamical evolution of the 3D model. Indeed, the AMR technique implies that the mesh of the simulation can vary in time and thus the amount of data being managed by MPI processes for both computation and I/O steps also varies. It introduces a lot of variability on I/O volume of each MPI process and it makes the I/O volume request complex.

Check-pointed data are stored in an Hercule **HProt** database and read back in memory upon restarting the simulation to resume the dynamical evolution of the model. Lighter but more frequent post-processing snapshots are stored by the RAMSES code in an Hercule **HDep** database. Those are meant to be read by post-processing softwares only (and never by RAMSES itself) for scientific analysis purposes.

Since it is beyond the scope of the IO-SEA project to redevelop another I/O scheme in RAMSES besides Hercule, and since Hercule will access the interfaces proposed within the context of the project, we plan to use the interfaces that will actually be implemented in Hercule (DASI, S3, ...). As of today, the Hercule library already uses an application-specific semantic approach to store data, so we expect the DASI interface implementation within Hercule will be quite straightforward.

6.2. RAMSES ephemeral services

- **NFS**
  The NFS interface will be used to read initial condition files (a few MB at the first running step in the session) and configuration files (a few kB at most) for each step, and write output log ASCII files (few MB). The output log files may need to be archived before ending the session. These data types do not represent a huge I/O load. These files will be accessed using standard POSIX I/O.
- **DASI**
  The DASI interface, can be used to write checkpoint data and read them back upon restart (HProt files). During checkpoints/restarts, the entire adaptive mesh refinement grid will be stored/read along with all the scalar quantities defined on that grid for a typical self-gravitating CFD 3D simulation. Together with this Eulerian description, a large catalog of Lagrangian particle information needs to be stored/read. We expect the major I/O contention to happen upon check-pointing/restarting the simulation since these are the most demanding steps in term of I/O load during our entire application workflow.

  The astrophysics-specific semantic approach of Hercule can easily be adapted to use the DASI interface, once it is implemented.

  The DASI interface will also be used to read/write reduced data sets for analysis purposes, namely the HDep files written by Hercule. These output files are designed to contain reduced information compared to a full HProt checkpoint, but they are meant to be produced at higher frequency, and reused more than once during the post-processing step ("in-situ" or post-mortem).

6.3. RAMSES workflow scheme

The RAMSES workflow can be split into two consecutive (but possibly overlapping) parts, first the dynamical evolution of a 3D model through time integration that will produce raw simulation 3D data at various timesteps (HProt checkpoints, only designed for restarts) and lighter subsets of chosen 3D

data for later scientific analysis (HDep backups). This will be the most intensive part of the workflow in terms of I/O, described in Subsection 6.3.1.

The second part of the workflow, which can be triggered as soon as some HDep backups are made available by the first workflow, is focused on scientific analysis of the data, possibly in an interactive way. This second workflow is described in Subsection 6.3.2.

6.3.1. RAMSES time integration workflow

The dynamical evolution of a 3D model within a RAMSES simulation leads to a quite linear dataflow. The first `initialisation` step of the workflow is designed to read initial condition files and a set of analytical models to set up the configuration of the 3D model at $t = 0.0$. During this step, one or several checkpoints are stored with the only purpose of restarting the simulation from these time steps (HProt database, DASI interface). Reduced descriptions of the 3D model are also stored (HDep databases, DASI interface), eventually at higher frequency, but for scientific analysis purposes only.

The `restart` step reads the latest HProt checkpoint produced by the previous (`initialisation` or `restart`) step and resumes the time integration of the 3D model. This step will enrich the HProt and HDep databases as well and can be repeated a user-defined number of times, until final simulation completion.

Both `initialisation` and `restart` steps can read/write configuration files (namelists) and write output log files that could eventually be stored for safekeeping before the end of the session. A proposed scheme for this workflow in the IO-SEA environment is described in the WDF in Listing 6.1 and command sequence in Listing 6.2.

Here we propose separate NFS ephemeral services for the initial condition data sets (read-only ASCII files used only once during the `initialisation` step), output log files (write-only ASCII files at low I/O bandwidth) and configuration files (negligible I/O operations on small ASCII files, with read/write permissions). In addition, two DASI ephemeral services are dedicated to intensive I/O output flows for checkpoint data and analysis backups, stored in HProt and HDep databases, respectively.

```
workflow:
    name: RAMSES_Workflow

services:
  - name: initial_conditions
    type: NFS
    attributes:
      namespace: ics
      flavor: medium
      mountpoint: /mnt/USER/Ramses_wf/{{ SESSION }}/ics

  - name: nml_configs
    type: NFS
    attributes:
      namespace: nmls
      flavor: medium
      mountpoint:/mnt/USER/Ramses_wf/{{ SESSION }}/nmls

  - name: logs
```

```
    type: NFS
    attributes:
      namespace: logs
      mountpoint:/mnt/USER/Ramses_wf/{{ SESSION }}/logs
      flavor: medium

  - name: checkpoints
    type: DASI
    attributes:
      configfile: /home/USERS/user_ramses/.dasi/dasiConfig_hprot.txt
      flavor: medium

  - name: hdeps
    type: DASI
    attributes:
      configfile: /home/USERS/user_ramses/.dasi/dasiConfig_hdep.txt
      flavor: medium

steps:
  - name: init
    location: cpu_module
    command: "srun -n {{ NPROCS }} ramses_job.sh
    services:
      - "initial_conditions" RO
      - "nml_configs" RO
      - "logs" WO
      - "checkpoints" WO
      - "hdeps" WO

  - name: restart
    location: cpu_module
    command: "srun -n {{ NPROCS }} ramses_job.sh
    services:
      - "nml_configs" RW
      - "logs" WO
      - "checkpoints" RW
      - "hdeps" WO
```

Listing 6.1: `ramses_workflow.yaml`, WDF for the RAMSES simulation workflow.

6.3.2. Scientific analysis workflow

As soon as the RAMSES simulation workflow starts populating the HDep database (depending on the user-defined output frequency), the scientific analysis workflow can be executed to reduce data, produce 2D projected images for visualisation purposes, compute profiles or spectrum, or even identify structures within the 3D model (e.g. stars, galaxies, discs, galaxy or stellar clusters). In typical workflows, very few types of data analysis treatment can be planned beforehand in a deterministic way. Most simulation workflows require a high level of interactivity to allow astrophysicists to do data exploration, physical model configuration optimisation or simply dynamical evolution monitoring.

```
session_date=‘date -u +%Y%m%d_%H%M%S‘
ramses_version=3.1
NRESTARTS=100 # Total number of restarts
NCORES=4096 # Total number of MPI processes

# Session name
session="Ramses${ramses_version}_sim_${session_date}"

# create namespaces for initial conditions, log files and configuration files
iosea-ns create --auto-create-dataset ics
iosea-ns create --auto-create-dataset nmls
iosea-ns create --auto-create-dataset logs
# create empty namespace/dataset for checkpoints/restarts
iosea-ns create --auto-create-dataset hprots
# create empty namespace/dataset for analysis-purposes backups
iosea-ns create --auto-create-dataset hdeps


# start a session for my workflow (described in the ramses_workflow.yaml file)
iosea-wf start WORKFLOW=ramses_workflow.yaml SESSION=$session

# run the workflow steps => first launch run from initial conditions
iosea-wf run SESSION=$session STEP=init NPROCS=$NCORES

# Restart simulation until final completion
for i in $(seq 1 $NRESTARTS) ; do
    iosea-wf run SESSION=$session STEP=restart NPROCS=$NCORES
done

# check status of jobs
iosea-wf status SESSION=$session

# stop session
iosea-wf stop SESSION=$session
```

Listing 6.2: IO-SEA interface commands to launch the RAMSES simulation workflow.

In the WDF in Listing 6.3 and the commands in Listing 6.4, we present an example of such a workflow where an interactive access is provided to the **hdeps** DASI ephemeral service containing the HDep databases before executing a custom scientific analysis Python script.

Depending on the final behaviour of the ephemeral service interface, we have considered that in our real production runs we may run the analysis in Listing 6.4 within the same session described by the WDF in Listing 6.1, in order to have simultaneous access to the hdeps data set. These issues will be clearer when we run on a prototype running the IO-SEA ephemeral services.

```yaml
workflow:
    name: ramses_analysis

services:
  - name: hdeps
    type: DASI
    attributes:
      configfile: /home/USERS/user_ramses/.dasi/dasiConfig_hdep.txt
      flavor: medium
```

Listing 6.3: `ramses_analysis_workflow.yaml`, WDF for the data analysis workflow.

```
#!/bin/bash

# start a workflow session named "analyze_hdeps"
iosea-wf start WORKFLOW=ramses_analysis_workflow.yaml SESSION=analyze_hdeps

# get interactive access to DASI HDep namespace/datasets
iosea-wf access SESSION=analyze_hdeps

# run analysis scripts on HDep backups (DASI interface)
my_astrophysical_analysis.py
```

Listing 6.4: IO-SEA interface commands to launch the RAMSES data scientific analysis workflow.

6.4. RAMSES summary

The previous sections introduced a preliminary scheme for the integration of the RAMSES astrophysical workflow in IO-SEA environment. We identified the WDF configuration and associated CLI, together with the NFS and DASI ephemeral services to be the best candidates for the RAMSES use case to bring significant advantages for our I/O scheme.

We keep in mind that the choices presented here might evolve in the future when each block of the IO-SEA software stack is going to reach maturity. At that point, we will be able to test a RAMSES simulation on prototype machines running IO-SEA services. After significant hands-on experience is obtained, we expect to converge on a set of workable RAMSES workflow templates that members of the RAMSES user community will be able to use in production for their own scientific projects.

7. Analysis of Electron Microscopy Images

7.1. Use-case overview

In this chapter we describe a strategy for IO-SEA services employed for streamlining management of the raw electron cryo-microscopy (cryo-EM) data. The workflow developed in this IO-SEA use case will provide a pipeline for automated annotation, publication, and archival of the cryo-EM data generated at CEITEC, Masaryk University (MU) electron microscopy facility. Our pipeline (Figure 4) will complement existing public databases, facilitating accessibility of the raw scientific data, optimizing mobilization and reuse of the raw cryo-EM data. The pipeline developed here will primarily focus on the cryo-EM data. However, once finalized, the pipeline will be generally applicable to the management of raw data generated by other biophysical or imaging techniques used in structural and cellular biology.

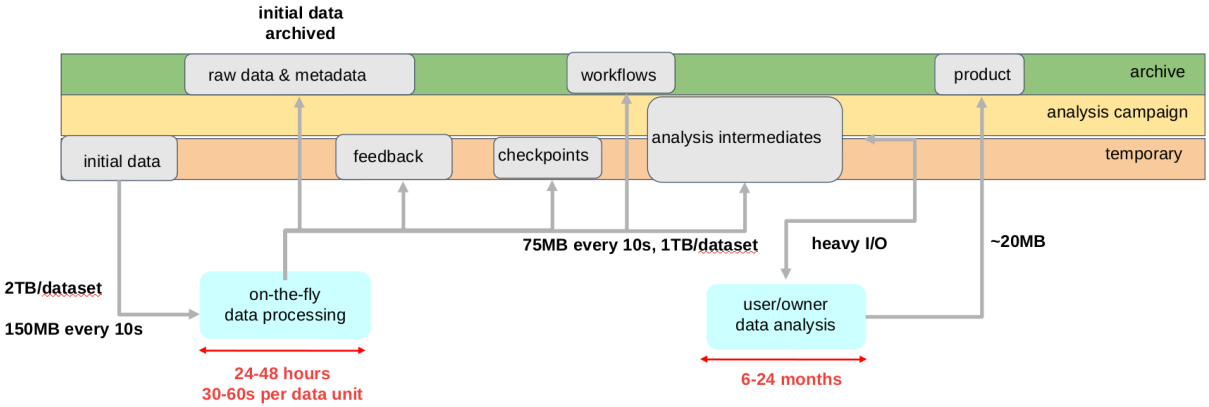

Figure 4: Electron microscopy data workflow diagram depicting the life-cycle of the cryo-EM data in the pipeline developed within the IO-SEA use case.

The pipeline developed in this use case will be based on establishing a federal cloud storage solution composed of a storage system situated close to the HPC center(s) or user computational resources for data analysis, and one or more data storage systems for preserving and archiving data before it is deposited in public databases. The data from the instrument will be synchronized on-the-fly to the HPC storage (part of the federal cloud) during data acquisition and supplemented with metadata, following a minimal data model oriented to enable automated data deposition to the public databases. The cloud service will annotate the data, assign unique identifiers, and add the data set into a catalogue. Eventually, the cloud service will make the data sets accessible once the embargo period (typically three years) expires, or submit the data to the public raw data repositories (EMPIAR in case of cryo-EM data) upon deposition of final analysis results.

7.2. Cryo EM ephemeral services

- **BB-NFS**
  The Burst-buffered NFS service will be used to expose a namespace in write mode for storing the results of the pipeline. As this part of the worklow is not generating a heavy I/O load, a simple NFS service could be sufficient. It will also hold the result data until they are uploaded to the tape archive, if the last step of the workflow is executed.
- **GBF**
  This service will be used to keep the TIFF images forming a single data set on a storage which can withstand a heavy I/O.

7.3. EM workflow scheme

The EM data flow can be split into multiple consecutive steps (Figure 5). First the data are captured by the instrument (electron microscope camera) and stored on the microscope computer storage (2-10TB SSD, RAID 0) connected to the camera via 10Gb/s optical cable. Next, the instrument operator will add an annotation, which cannot be extracted from the images, and initiate submission of the data to the cloud together with a workflow for data analysis. The information read from the data together with the annotation provided by the operator forms a minimal data model that sufficiently describes the raw data to make it discoverable after the publication. It contains information about laboratory generating the data, the owner of the data, the imaged sample, and the measurements parameters.

Lifecycle of the real-time processing service will be managed outside of the IO-SEA stack, the most important integration will be the batch processing case. In this case, similar pipeline will be used with different parameter values. Once the imaging is complete, the entire raw data set is stored in a storage close to the HPC cluster used for the batch processing.

Integration with the IO-SEA HSM will enable the user to transparently load the data set on a fast flash-based storage and execute the individual steps of the processing pipeline, including the last one, the upload and publication of the data on public websites.

Subsequently, the allocated HPC resources will read the data analysis configuration file and initiate the on-the-fly data processing. Once the acquisition of the data is terminated, data owner is notified via an email which contains information about the data location.

7.4. EM ephemeral services

Selection of the ephemeral services in this use case assumes that the raw data sets corresponding to individual imaging campaigns are available in the IO-SEA stack as data sets.

Raw data obtained from the microscopes are transferred to the HSM by a service running outside of the IO-SEA stack and its life-cycle is managed by the operations front-end. Components of the

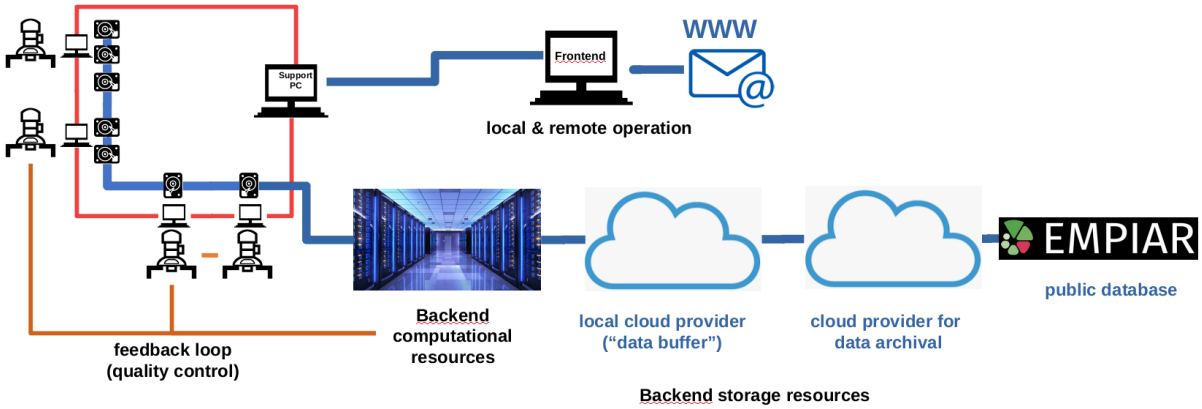

Figure 5: General strategy of the cryo-EM data management pipeline. Data generated on the microscope are automatically submitted to the IRODS federated cloud and automatically go pass into on-the-fly data analysis pipeline on the remote HPC resources.

service run on the CEITEC infrastructure as well as on the IT4I virtualization infrastructure, handling the data transfer to the HPC clusters storage.

The processing pipeline consists of multiple GPU-enabled applications, each run processing a single TIFF image of the data set. As a single data set consists of around 15 thousand files which have to be loaded in the GPU memory, heavy I/O is expected. The output from the individual processing stages is stored by a common ephemeral service, from where it gets transferred to long-term storage (tapes, HDD cluster). Each step processes the entire data set by the single application and stores the result on the common service. Users can then check the result of the processing and adjust the parameters of the next step as necessary.

```
workflow:
  name: CryoEM_batch

services:
  - name: cryoem_dataset
    type: GBF
    attributes:
      namespace: {{ DATASET }}
      mountpoint: /mnt/USER/CryoEM_batch_{{ SESSION }}
      Size: 8TB
      flavor: medium
      datanodes: 4
    datamovers:
      - name: datamover_raw
        trigger: step_start
        target: flash
        operation: copy
        elements:
          - "dataset/*.tiff"

  - name: cryoem_results
    type: BB-NFS
    attributes:
```

```
      namespace: cryoem_results_{{ SESSION }}_{{ DATE }}
      mountpoint: /mnt/USER/CryoEM_results_{{ SESSION }}
      size: 2TB
      flavor: medium
      datanodes: 4
    datamovers:
      - name: datamover_results
        trigger: step_stop
        target: tape
        operation: move
        elements:
          - "*"

steps:
  - name: dataprepare
    location:
      - datanodes
    command: "sbatch data_download.sh"
    services:
      - name: cryoem_dataset
        datamovers:
          - datamover_raw

  - name: MotionCor2
    location:
      - gpu_module
    command: "sbatch run_motioncor2.sh"
    services:
      - name: cryoem_dataset
      - name: cryoem_results

  - name: GCTF
    location:
      - gpu_module
    command: "sbatch run_gctf.sh"
    services:
      - name: cryoem_dataset
      - name: cryoem_results

  - name: dataupload
    location:
      - datanodes
    command: "sbatch data_upload.sh"
    services:
      - name: cryoem_results
        datamovers:
          - datamover_results
```

Listing 7.1: `cryo_wdf.yaml`, WDF for the CryoEM batch processing workflow.

The workflow description file shown in Listing 7.1 shows the initial version of the pipeline with two applications executed sequentially. The data are moved between the storage tiers by the datamover services in dedicated workflow steps at the beginning and at the end of the workflow. This pattern can be easily reused as more applications will become available in the pipeline. Sequence of commands

from the IO-SEA command line which can be used to launch an instance of this batch workflow is described in Listing Listing 7.2.

```
sample_id=sampleX
session_date=‘date -u +%Y%m%d_%H%M%S‘

# Create a dataset for storing the batch analysis result
iosea-ns create --auto-create-dataset cryoem_results_$sample_id_$session_date

# Create a session
iosea-wf start WORKFLOW=cryo_wdf.yml SESSION=$sample_id DATE=$session_date DATASET=<My_Dataset>

# Trigger each workflow step, one after each other when previous one is terminated
iosea-wf run SESSION=$sample_id STEP=dataprepare
iosea-wf run SESSION=$sample_id STEP=MotionCor2
iosea-wf run SESSION=$sample_id STEP=GCTF
iosea-wf run SESSION=$sample_id STEP=dataupload


# Stop the session
iosea-wf stop SESSION=$sample_id
```

Listing 7.2: `cryo_batch.sh`, shell script with IO-SEA commands used to run the cryo-EM workflow.

7.5. EM Summary

We have described the workflow for the cryo-EM image processing which will be implemented in two forms. The first one is orchestration of the real-time processing service triggered by the front-end application used by the microscope operator. The lifecycle of this service will be managed out of the IO-SEA HSM stack scope.

The later form of the workflow is batch processing of the entire cryo-EM data sets, where the integration with the IO-SEA HSM is crucial to provide sufficient performance. We have described this case in a workflow description file in Listing 7.1. As well as provided the list of ephemeral services used in this case. The user of this workflow will then use the IO-SEA commands to launch the individual stages of the pipeline manually up to the last stage, where the data are uploaded to a long term storage. The pipeline described here is in initial stages and more steps will be added later during the project, as necessary.

8. Weather forecasting workflow

8.1. Use-case overview

In this chapter we describe the IO-SEA ephemeral services likely to be used during a run of the weather forecasting workflow. In previous deliverable reports [5, 7] we described the workflow and its two principal steps, as illustrated in Figure 6.

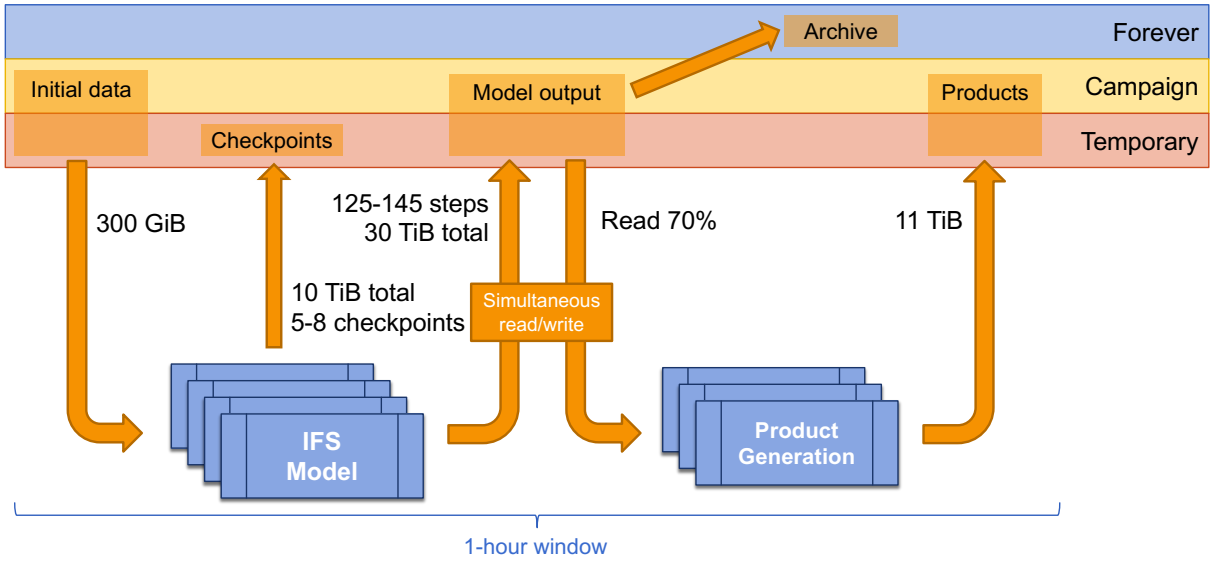

Figure 6: Weather forecasting data flow. One instance of the IFS model is run at high resolution and 51 instances are run as an ensemble at coarser resolution. Each step of the high-resolution run and each common step of the ensemble trigger one product generation run.

This workflow is fully automated, and the operational execution is controlled by a workflow manager that schedules the steps based on their dependencies. First, all instances of the ensemble and high-resolution IFS models are submitted to the HPC system. The model uses some read-only data in the form of both configuration files and initial state, and writes regular checkpoints as well as a set of snapshots which are the principal data set in the workflow, later referred to as *model output*. When the data associated with one time step have been produced by all relevant model instances — that is, either the single high-resolution run or all the coarser-resolution ensemble members, the associated product generation step is triggered, reading most of the model output for that time step and producing *products* extracted according to a requirements file.

Since the workflow has to run within a strict one-hour time-critical window, it is crucial to process the model output as soon as possible. The product generation steps read slices of data across the output of all of the model instances. This causes significant access contention on the filesystem, putting severe correctness and performance constraints on the underlying storage if all of the read and write operations are to be satisfied simultaneously.

In operational use, once the model and product generation steps finish, the model output is archived into ECMWF's meteorological archive (MARS), and stays on the parallel filesystem for a few days to allow usage by downstream processing and evaluation. The products are immediately disseminated to ECMWF member states and private customers. These steps are not required in the benchmarking workflows envisaged in the project.

8.2. Weather forecasting ephemeral services

- **NFS**
  The NFS interface will be the main interface for ancillary data sets like configuration files and read-only inputs. They will be accessed using standard POSIX I/O, and do not represent a significant amount of I/O load.
- **DASI**
  The DASI will be at a central position in the workflow. It will receive the output from all the ensemble members, and make it available to the product generation jobs. Significant read-write contention is expected here. The applications already use semantic data access by means of a domain-specific language, which means they can easily be adapted to use the DASI.
- **GBF**
  Temporary files such as logs and checkpoints will be placed into a GBF data set since they are meaningful only during the session. Any file that may keep value after the run will be archived before ending the session.

8.3. Sample weather forecasting workflow scheme

The workflow can be described according to the Workflow Description File (WDF) shown in Listing 8.1. This listing contains the two primary workflow steps described above, namely the forecast model step (`ifs`) and the product generation step (`pgen`).

The workflow components are parametrised according to the base date and time of the forecast, that is, the valid time for the input (observational) data. Further, the forecast model runs as an ensemble of lightly perturbed forecasts. As such the forecast step is parametrised to indicate which member of the ensemble should be executed. Forecasts produce output at a range of timesteps, indicating the number of hours into the future that are being forecasted. The post-processing steps are run per output step, and as such are parametrised for this.

The WDF indicates the input and output locations used by each of the model steps. Configuration and input data, which are both small and are not modified during the forecast run, are stored on NFS storage. Checkpoint information and log output, which are large in volume and produced in a time-critical fashion, but whose useful lifetime is very constrained are stored in temporary storage. Final output data is placed into externally visible storage where it can be accessed for dissemination.

Finally, the most significant data to consider (by volume, number, stress on the storage subsystem and timeliness requirements) are the model output data which will be reprocessed by post-processing. This data will be stored in the high-performance semantic object store, DASI.

```yaml
workflow:
  name: ECMWF_prod

services:
  - name: configs
    type: NFS
    attributes:
      namespace: "configs-{{ version }}"
      mountpoint: /media/prod/configs
      flavor: medium
  - name: ifs-input
    type: NFS
    attributes:
      namespace: "input-{{ version }}-{{ date }}{{ time }}"
      mountpoint: /media/prod/input
      flavor: medium
  - name: ifs-checkpoints
    type: GBF
    attributes:
      mountpoint: /media/prod/checkpoints
      size: 12TB
      flavor: medium
      datanodes: 4
  - name: logs
    type: GBF
    attributes:
      mountpoint: /media/prod/logs
      size: 10GB
      flavor: medium
      datanodes: 4
  - name: ifs-outputs
    type: DASI
    attributes:
      configfile: "/home/prod/{{ version }}/dasi.yaml"
      flavor: medium
      datanodes: 20
  - name: products
    type: NFS
    attributes:
      namespace: "products-{{ date }}{{ time }}"
      mountpoint: /media/prod/products
      flavor: medium
      datanodes: 4

steps:
  - name: ifs
    location: cpu_module
    command: "sbatch /home/prod/{{ version }}/{{ date }}{{ time }}/ifs_{{ member }}.sh"
    services:
      - configs
      - ifs-input
      - ifs-checkpoints
      - logs
      - ifs-outputs

  - name: pgen
    location: cpu_module
    command: "sbatch /home/prod/{{ version }}/{{ date }}{{ time }}/pgen_{{ model_step }}.sh"
    services:
      - configs
      - ifs-outputs
      - logs
      - products
```

Listing 8.1: `ecmwf_workflow.yaml`, WDF for the weather forecasting workflow.

Listing 8.2 shows how the ECMWF workflow should be run. In particular, it demonstrates how the forecast should be parametrised, and how the parametrised components should be launched. Note that it skips across any of the workflow notification and control required to ensure that the post-processing steps are run after the data is available. In the context of Work Package 1, this will be ensured by using the Kronos workflow management package.

```
# Set run parameters
date=20220517
time=1200
version=47.3
ensemble=50
nsteps=100

# Start workflow
session="prod_${date}_${time}"
iosea-wf start WORKFLOW=ecmwf_workflow.yaml SESSION=$session date=$date time=$time
     version=$version

# Run ensemble steps
for i in $(seq 1 $ensemble) ; do
    iosea-wf run SESSION=$session STEP=ifs member=$i
done

# Run product generation
# NOTE: this would be run by a workflow manager upon notification
for i in $(seq 1 $nsteps) ; do
    iosea-wf run SESSION=$session STEP=pgen model_step=$i
done

# Stop the session
iosea-wf stop SESSION=$session
```

Listing 8.2: IO-SEA interface commands to launch the workflow.

8.4. Weather forecasting summary

We have defined the weather forecasting workflow in terms of ephemeral services and storage requirements. The workflow comprises an ensemble run of the IFS forecasting model and a product generation step for each timestep of the ensemble model. The workflow will be orchestrated using Kronos, which will interact with the IO-SEA workflow handler described in Chapter 3 and trigger the production generation runs upon notification from the ensemble steps. The critical data set that hosts the model output will be accessed using the DASI, while ancillary data sets will use NFS and

GBF. We expect this workflow to provide a good stress test of the ephemeral services as well of the software supporting the DASI data set.

9. Multi-physics regional Earth system model

9.1. Use-case overview

In this chapter, we describe the IO-SEA ephemeral services likely to be used during a run of the Terrestrial Systems Modelling Platform, (TSMP) workflow. In previous deliverable reports [5, 7] we described the workflow as illustrated in the workflow diagram in Figure 7. This is a typical workflow used in current production jobs and the larger jobs in the foreseeable future. Within IO-SEA, TSMP is working towards efficiently handling the increase in I/O and data movements spawning from upcoming higher resolution terrestrial system simulations for which ephemeral services are a prime tool.

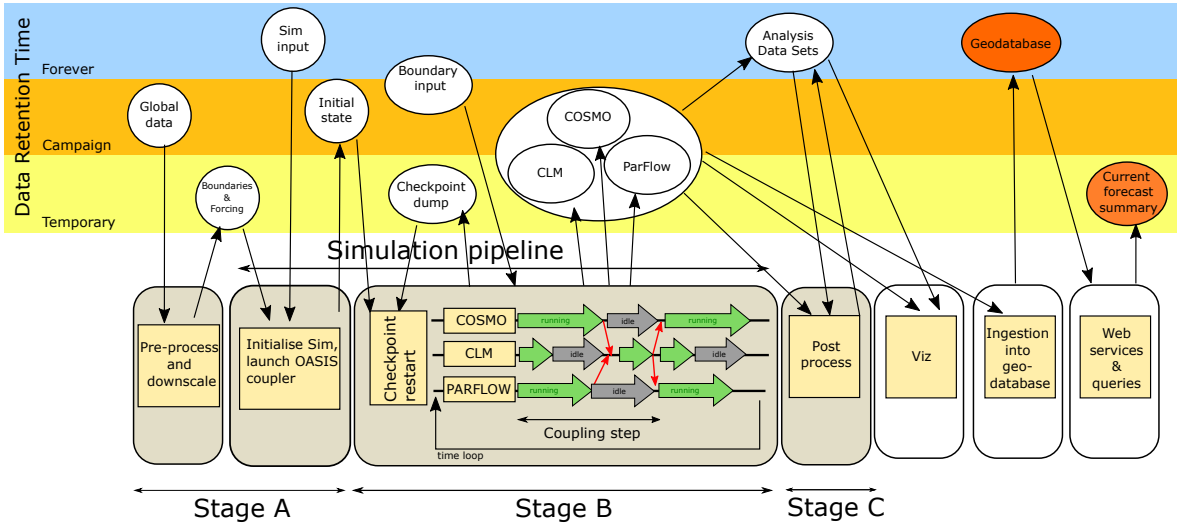

Figure 7: TSMP IO Workflow

The workflow in Figure 7 illustrates the core of the simulation pipeline, including the three coupled core model components: COSMO (v5.01) model for atmospheric simulations, CLM (v3.5) land surface model, and ParFlow (v3.9/v3.10) hydrological model. Pre/post processing and visualization steps may be considered, but are not strictly necessary. The interest in pre/post processing steps is that they imply potentially intensive I/O operations. Consequently, the proposed data nodes and ephemeral services may be very effective in reducing the costs of data movements and facilitate in-situ processing of simulation output. The outputs of the three component models are independent of each other, both in their generation (i.e., different binaries are generating the output) and in how they are written (i.e., different file sets). However, due to the coupled physics that they represent (complex interactions and feedbacks of mass, energy, and momentum across the land surface, subsurface, and atmosphere) in TSMP, the choices for namespace and data set definitions are specified in a way that allows to maintain both the independence and the interdependence of the data. That is, the TSMP dataset comprises all of the results, but namespaces allow to expose subsets corresponding to the different components.

Besides the static files, we also have input data sets used for configuring initial conditions such as land surface in the CLM component or initial-boundary in ParFlow. Therefore, step *A* is the pre-processing stage to set up initial conditions (initialization) and to prepare (interpolation/projection) boundary forcing data (in particular for the atmospheric component). In step *B*, the coupled simulation takes place, with coupling occurring through MPI via OASIS coupler. In this step, we are considering nested namespaces or intersections for each component individually. The output from the component models may or may not be concurrent, and may or may not require dedicated data nodes. The final step *C* can include different post-processing procedures, including visualization, analytics, pushing to online publication, etc. It is foreseen that some (if not all) of step *C* could run concurrently with step *B*, heavily leveraging on data nodes.

9.2. TSMP ephemeral services

- **NFS**
  The NFS interface can be used in TSMP for workflows that require POSIX-compliant file shares and for IO and hierarchical storage during data collection from different component models outputs while they are running on the different computational nodes. This would be the most straightforward continuation of current TSMP workflows, which may be convenient for certain use cases. It will also likely be convenient when retrieving data sets used for input or comparisons, which may not exist within the IO-SEA solutions. In such cases, the use of datamovers will be relevant, for example, to store/retrieve data subsets from long-term storage, for staging of input data, or post processing.
- **DASI**
  TSMP's native POSIX output is planned to be ingested into DASI. DASI will provide an application which would allow the raw (POSIX) TSMP output to be then pushed into DASI. It is important to highlight that, because TSMP's I/O is handled by each component model (whose development is not controlled by TSMP), it is not foreseen to implement the DASI API within the TSMP component models.
- **SBF & GBF**
  Increasing model resolution for coupled land-atmosphere simulations in TSMP logically causes an increase in I/O and storage volumes of output per model. Managing both I/O and storage of this increased volume with the help of GBF services is a potential solution which we will explore in IO-SEA. Additionally, GBF can be used together with with a POSIX-to-DASI application. One of the key uses of GBF is to write TSMP's output into a GBF ephemeral POSIX system (e.g., flash memory on data nodes), and make use of data nodes to ingest the POSIX output into DASI on-the-fly.

  For the pre-processing step, a mounted single node of the SBF ephemeral service might be sufficient. These resources would be used to stage input data which may be pulled from the storage tier (from S3 objects, DASI, or POSIX), albeit requiring a datamover to retrieve it, or which may need to be fetched from remote sources (e.g., global scale model output). Optimal solutions for these purposes will require experimentation.

9.3. Sample TSMP workflow scheme

Herein, we assume TSMP runs are performed with the fully coupled system, including COSMO, CLM, and ParFlow, which is the most general case, despite TSMP allowing for subsets of coupled models (for which the template developed here is quite applicable). We outline below an example scheme for running a TSMP workflow in the IO-SEA ecosystem, divided into three stages.

9.3.1. Stage *A*: TSMP Pre-processing

Input data sets for one of the typical test case , the so-called EUR-CORDEX test case (full name: EUR-11 pan-European CORDEX case, a integrated simulation over the entire European continent) include static files, initial and boundary conditions, and geometry coupling files, requiring in the order of tens of gigabytes. Some intermediate preprocessing output can be produced. This stage can also include retrieving data sets from remote sources or lower levels of the storage system. Therefore, it will likely benefit from datamovers and from the possibility of launching these preprocessing stages somewhat in advance to actually launching computations (Stage B).

9.3.2. Stage *B*: TSMP simulation pipeline

The simulation pipeline includes the following phases for each of the components:

- Model initialization and definition which includes model grid, model partition and I/O, constructing the coupled system via the OASIS layer which handles the coupling sequence and frequency, the names of the coupling fields, the spatial grid of the coupling fields, and finally the type of transformations of the 2D coupled fields.
- Solving the systems of partial differential equations.
- Sending–receiving of coupling fields between components through MPI communications.
- Checkpointing and restarts of simulation (due to reaching maximum job duration as enforced by the scheduler).
- Termination of simulation of each component model.

Stage *B* includes writing output from each of the model system components into a POSIX file system. As illustrated in Figure 8, we imagine a namespace `tsmp_eu_cordex_run_20220201` which could expose all of the input and output data for TSMP. We also imagine component-specific namespaces (e.g., `tsmp_eu_cordex_run_20220201_parflow` for ParFlow), which only expose the data relevant for a single component. This approach is useful for several purposes, such as (i) building ensemble experiments which only partially reuse component input, (ii) a fine-grained ingestion into DASI, (iii) postprocessing workflows which may only be interested in the output of a single component, (iv) visualization of selected output data sets.

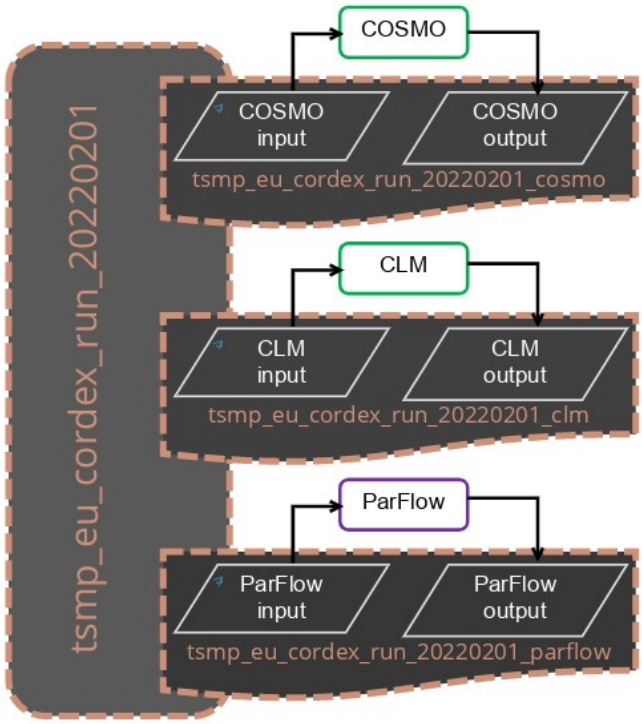

Figure 8: TSMP namespaces and their relationship to the component models.

9.3.3. Stage *C*: TSMP post-processing

Post-processing workflows for TSMP can be very varied. In current applications it is often necessary to extract single fields from the sophisticated NetCDF files, or perform output conversion before analytics or visualization. We of course do not attempt to encompass all possibilities here. Consequently, we attempt to abstract these possibilities into a generic post-processing workflow, as illustrated in Figure 9. Moreover, we envision that Stage *C* may be concurrent with Stage *B*, (in situ processing), rather than sequentially after Stage *B* (post processing). However, we refrain from selecting a particular solution for this at this stage, as it is likely that full flexibility is appreciated by TSMP users, depending on the complexity of the analytics and visualizations required. For both of these possibilities, it is relevant to distinguish the output of each of the component models, as they significantly differ in size, complexity and content, which pragmatically means that each of them typically requires different post processing workflows. It is also of course very likely that all three are required for particular analytics. Finally, Stage *C* could be performed based on the DASI-ingested data, or on the POSIX TSMP output. Presumably, the choice will depend on the complexity and computational cost of the post processing workflows, and will be linked to the aforementioned concurrent or sequential relationship between Stages *B* and *C*. Therefore, in this stage heavy use of the POSIX-to-DASI and DASI-to-POSIX applications are foreseeable.

9.4. TSMP Summary

We have defined the TSMP workflow template in a *YAML* file, and also imported and exported the workflow template *YAML* file in a session called (for example purposes only) CORDEX, which references a typical simulation over the CORDEX EUR-11 pan-European domain.

The workflow instantiates a session spanning all three stages (*A,B,C*) shown in Figure 7. Listing 9.1 shows the definition of three different ephemeral services.

The first service (`ES_preprocessing`) is intended for fetching and pre-processing data to create input files for the simulation, and clearly relates to Stage *A*.

We also define a set of three ephemeral services (`ES_cosmo`, `ES_clm`, `ES_parflow`) imagined as GBF services to receive the POSIX output from each of the component models. The POSIX output is imagined to be received into very fast storage hardware (e.g., flash memory). The storage sizes indicated in the YAML file are for example purposes only, and of course need to be properly evaluated. These services will be used during the main simulation pipeline (Stage *B*), to temporarily store the raw output and also to ingest it into DASI.

The `ES_DASI` ephemeral service will be used for the ingestion of the POSIX output into DASI. This can be imagined to occur on-the-fly during Stage *B* (i.e., immediately after output is generated by TSMP component models), or afterwards, during some post-processing (Stage *C*). In the steps in our sample YAML file we assume the Stage *B* case.

It is expected for all of these ephemeral services to run in a single session. Several steps are therefore designed. The first step (`step_preprocessing`) fully encompasses Stage *A*, and would typically be run on CPUs, given the heavy interactions with the file system and network (possibly with remote locations).

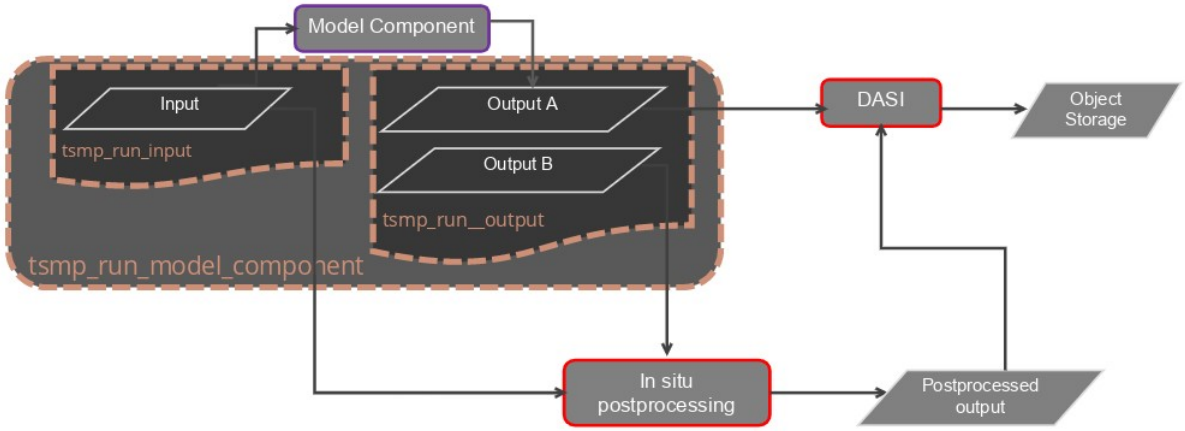

Figure 9: Stage *C* generic post-processing workflow, exposing input and output namespaces, and their interaction with DASI.

Stage *B* is comprised of two steps. `step_simulation` comprises the simulation supported by the individual ephemeral services for each component model. Notably, `step_simulation` specifies two locations, which enables for TSMP to launch a modular job on the MSA system. Within this step, many sequential jobs may be requested to the scheduler, as TSMP runs usually exceed maximum job time. This is assumed to be handled in the `sbatch` script. It it is important to highlight that the namespace exposed by `ES_preprocessing` is required in this stage as read-only. The second step within Stage *B* is within `step_ingestion`. Ingestion of the POSIX output into DASI (exposed by `ES_DASI`) would occur here. These two steps (`step_simulation` and `step_ingestion`) are imagined to be able to run concurrently (i.e., on-the-fly DASI ingestion) or sequentially. Another relevant issue in this step is checkpointing and restarting. A temporary location for checkpoint files and data input for restarting TSMP on a new scheduled job is required. We assume here that the required data will survive in the GBF ephemeral services. However, we acknowledge that there might be a need to actually store such data in the tiered storage system, possibly via an NFS service.

Finally, `step_postprocessing` launches a post processing, which may take POSIX output, process it and push it into DASI, or also pull data from DASI, process it, and return it to DASI. We do not specify any particular choice here.

```
workflow:
  - name: tsmp_eu_cordex

services:
  # for fetching data and pre-processing
  - name: ES_preprocessing
    type: GBF
    attributes:
      namespace: tsmp_eu_cordex_pre_run_20220201
      mountpoint: "/mnt/USER/tsmp_eu_cordex/{{ SESSION }}/ES_PreP"
      flavor: medium
      size: 2TB
      datamovers:
        - name: datamover1
          trigger: step_start
          target: flash
          operation: copy
          elements:
            - "cosmo_input"
            - "clm_input"
            - "parflow_input"

  # for the main simulation pipeline and its POSIX output
  - name: ES_cosmo
    type:GBF
    attributes:
      mountpoint: "/mnt/USER/tsmp_eu_cordex/{{ SESSION }}/cosmo"
      datanodes: 2
      flavor: medium
      size: 4TB

  - name: ES_clm
    type:GBF
    attributes:
      mountpoint: "/mnt/USER/tsmp_eu_cordex/{{ SESSION }}/clm"
```

```
      datanodes: 1
      flavor: medium
      size: 4TB

  - name: ES_parflow
    type:GBF
    attributes:
      mountpoint: "/mnt/USER/tsmp_eu_cordex/{{ SESSION }}/parflow"
      datanodes: 2
      flavor: medium
      size: 4TB

  # to ingest POSIX output into DASI
  - name: ES_DASI
    type:DASI
    attributes:
      configfile: "/home/user1/.dasi/dasiConfig.txt"

steps:
  - name: step_preprocessing
    command: "sbatch tsmp_preprocessing"
    location: cpu_module
    services:
      - name: ES_preprocessing

# this step runs a modular TSMP job. Concurrency of the different component models is controlled
    within the sbatch script.
  - name: step_simulation
    command: "sbatch tsmp_launch"
    location:
      - cpu_module
      - gpu_module
    services:
      - name: ES_cosmo
      - name: ES_clm
      - name: ES_parflow
      - name: ES_preprocessing RO

# to push posix output into DASI, should run concurrently (although not strictly) with the
    previous step
  - name: step_ingestion
    command: "sbatch tsmp2dasi"
    location: datanodes
    services:
      - name: ES_DASI RW
      - name: ES_cosmo RO
      - name: ES_clm RO
      - name: ES_parflow RO

# could run sequentially after the two previous steps, or perhaps also concurrently
  - name: step_postprocessing
    command: "sbatch tsmp_postprocessing"
    location: datanodes
    services:
      - name: ES_cosmo RO
      - name: ES_clm RO
      - name: ES_parflow RO
```

```
      - name: ES_DASI RW
```

Listing 9.1: Workflow Description File (WDF) for a typical TSMP workflow.

```
#!/bin/bash

# create namespaces & datasets
iosea-ns create --auto-create-dataset tsmp_eu_cordex_pre_run_20220201

# start session
#session name : Cordex_Session

iosea-wf start WORKFLOW=TSMP_workfkow.yaml SESSION=Cordex_Session

# Run Step A
iosea-wf run SESSION=Cordex_Session STEP=step_preprocessing

# Run main simulation
iosea-wf run SESSION=Cordex_Session STEP=step_simulation

# run to ingest POSIX into DASI
iosea-wf run SESSION=Cordex_Session STEP=step_ingestion

# Run Step C
iosea-wf run SESSION=Cordex_Session STEP=step_postprocessing

# display status
iosea-wf status SESSION=Cordex_Session

# finish workflow
iosea-wf stop SESSION=Cordex_Session
```

Listing 9.2: Script to launch the TSMP IO-SEA workflow

10. Lattice quantum-chromodynamics

10.1. Use-case overview

In this chapter we describe the IO-SEA ephemeral services likely to be used during a run of the Lattice–Quantum-ChromoDynamics (LQCD) workflow. In previous deliverable reports [5, 7] we described the workflow in four steps as illustrated in the workflow diagram in Figure 10.

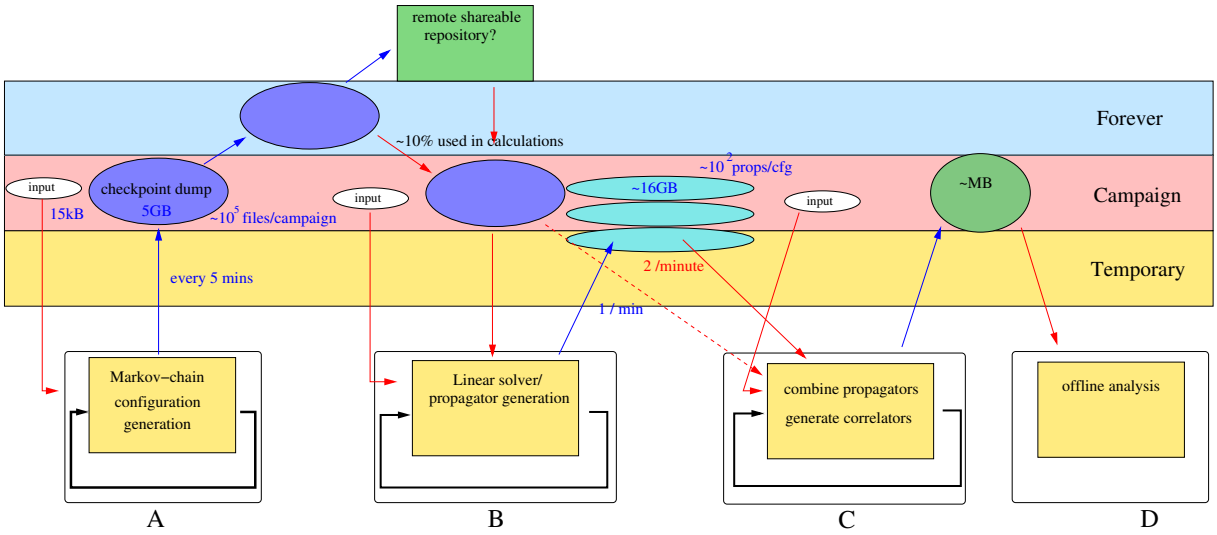

Figure 10: LQCD workflow diagram, showing data movement through the storage hierarchy for four workflow steps. step *A* is gauge configuration generation, and *B* – *D* are steps of the measurement phase. Purple ovals represent gauge configurations, cyan ovals are quark propagators, and the green oval represents hadron correlators.

Step *A* begins with a single input file defining the physical and algorithmic parameters. The step generates a Markov chain of gauge field configuration files through a hybrid Monte Carlo (HMC) algorithm. The HMC algorithm proposes updates to the current gauge configuration, which are accepted or rejected in a Metropolis test. Regardless of whether the update is accepted or rejected, the configuration is check-pointed as a gauge field file. (It is possible two successive checkpoints may be identical, in the case of Metropolis test rejection).

A single compute job produces a number of gauge field files (subject to wall-clock constraints). Along with each gauge field file, an input file is produced which could be used to restart the next HMC job.

In step *B*, a subset of the gauge files is processed, producing *propagator files*, which contain solution vectors to large, sparse linear systems defined by the gauge field. In step *C*, the propagators from a given gauge field are combined in *contractions* (element-wise products) to give smaller arrays called *hadron correlators*. Finally, in offline analysis, physics results are extracted from the set of hadron correlators.

It is important to note that the full workflow is never executed in an automated, linear fashion. At the beginning of a project one does not know the full number of gauge field configurations needed to

achieve an acceptable level of statistical uncertainty on the final results. Soon after the start of the gauge configuration generation in step *A*, some analysis must begin to determine if the configurations are 'thermalised' (if some measurable quantities have reached an equilibrium value), and to determine the auto-correlation time (roughly how many updates must be performed before we reach a statistically independent configuration). These quantities indicate at what point we can begin the calculations in Steps *B* and *C*, and the number of gauge files to skip between independent runs of steps *B* and *C*.

Furthermore, peeking at the timestamps of the accumulating gauge field files in step *A* allows the diagnosis of performance problems. Finally, extracting the Metropolis accept/reject rate from the log files is important for tuning the algorithm. (An accept rate of ~70% is considered optimal.)

After a number of de-correlated, thermalized configurations are generated, step *B* — *D* calculations commence with step *A* continuing. Producing draft results on-the-fly gives further chances to identify algorithmic mistakes or to restart with better-tuned physics parameters. Only when early results with statistical uncertainties are available from step *D* can we begin to estimate the number of gauge configuration files to demand from step *A* in order to achieve acceptable precision on the final result.

As we consider in Section 10.2 below how to port the LQCD workflow to the IO-SEA ecosystem and what services to use, we must take into account all of these patterns of access and data use.

10.2. LQCD ephemeral services

- **NFS**
  In the LQCD workflow, the dominant I/O channel will be through the NFS interface. The application code is the widely-used CHROMA [8], which is designed to handle POSIX paths to input and output files. It is beyond the scope of the IO-SEA project to re-design the I/O scheme of CHROMA. In the proposed LQCD workflow scheme below in Section 10.3 we focus exclusively on NFS services. BB-NFS services will be considered later if the performance of a single NFS server is not sufficient.
- **BB-NFS**
  Before actual testing on a prototype machine, it is difficult to say how much benefit may be gained from the ephemeral services NFS interface. We may also test with the BB-NFS, but will not detail this possibility in the example schemes in Section 10.3.
- **DASI**
  The DASI interface represents a novel way of organising scientific data, and could certainly be applied to the storage of, e.g., gauge field configuration files. However its direct use in the CHROMA application would require a major re-write of not only CHROMA, but the QIO library which it uses. For technical and time reasons, this level of re-coding is unfeasible at this time.

  However, the stand-alone DASI-to-POSIX application is intriguing, and we plan tests as a data archive system before and after the execution of CHROMA.

10.3. Sample LQCD workflow scheme

We have not yet identified a single best way to use the suite of IO-SEA ephemeral services in the LQCD workflow. It is likely that the optimal scheme is dependent on many factors, including the size of the specific problem, number of files generated, and rate and pattern of access in a given physics project. In internal discussions we have identified a number of possible schemes, indicating that the IO-SEA solution will allow considerable flexibility to the LQCD user.

Important factors in designing the scheme are:

- Not every gauge file produced in step *A* is processed in *B* and *C*. Typically every tenth checkpoint is used.
- Different step *B* jobs are independent and run concurrently.
- For a given gauge field file, the step *C* (contraction step) can only commence after the propagators from step *B* are written out.
- The contraction step *sometimes* requires the gauge file.
- The propagators files have less long-term usefulness than the gauge files.
- having fewer and larger data sets introduces some convenience to data management and archiving.

Clearly, the simplest scheme is to have a global data set containing all data files for input or output. In this example, the global data set might be attached to a single workflow session and all jobs for steps *A* — *D* are run within that session. An advantage of this scheme is that it is simple to enact and presents a low barrier to entry for novices of the IO-SEA environment.

In this section we describe a scheme of moderate complexity, that separates different data types into different data sets, with the idea that one might manage gauge files, propagators and hadron correlators differently as they have different useful lifetimes. This slightly more complex scheme allows us to explore more features of the IO-SEA interfaces.

We outline below an example scheme for running an LQCD workflow in the IO-SEA ecosystem, broken up into workflow steps. Figure 11 provides an overview of the relation between file types and compute jobs, and how we might group the data into data sets.

For readability and organization we separate the steps into three workflow sessions in this example. We include an important note at the end describing how they might be combined to provided flexibility to running the workflow.

10.3.1. Step *A*: HMC gauge configuration generation

As described above in Section 10.1, step *A* is the generation of an ensemble of gauge field configuration files. This ensemble is a Markov chain, meaning that each new gauge file is generated by updates to the previous one. The initial run begins with parameters from an XML input file. A compute job can typically generate several dozen updates within the wall-clock limit, each of which is check-pointed with a corresponding XML file serving as an input file allowing restart from that point.

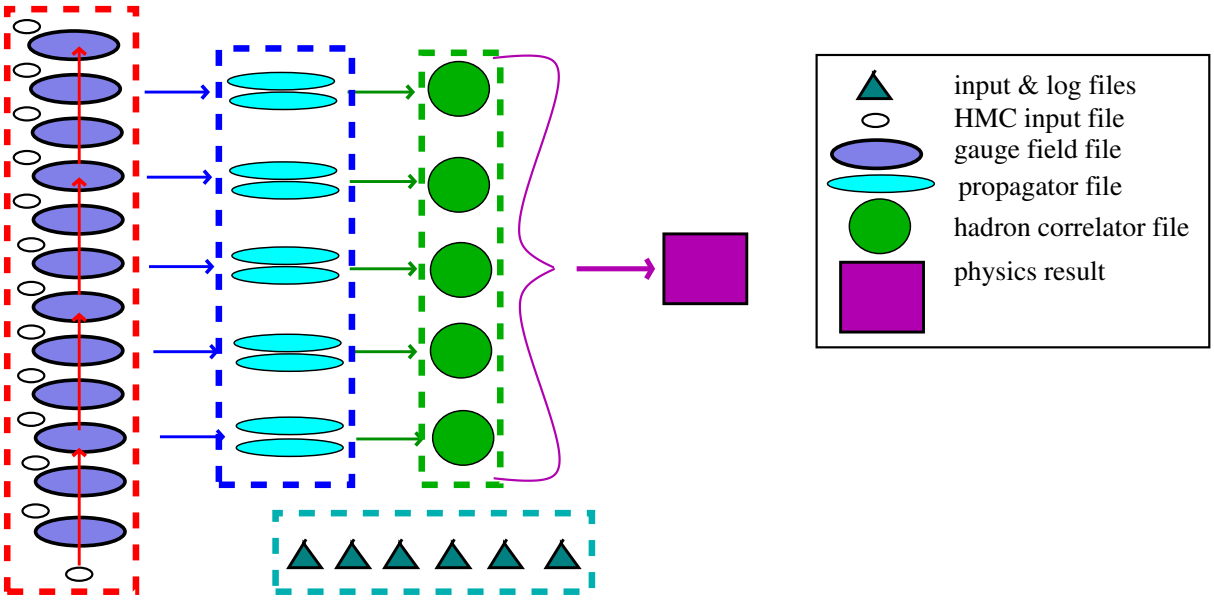

Figure 11: LQCD workflow data dependence. Red arrows represent step *A* HMC jobs producing gauge files (purple ovals) and restart input files (small white ovals). Step *B* solver jobs (blue arrows) each produce a set of propagators from a given gauge file. Step *C* contraction jobs (green arrows) combine propagators to produce hadron correlators (green ovals). Finally, in step *D*, offline analysis of the set of hadron correlators, produces a physics result (purple square).

In Figure 11, the red arrows represent individual step *A* HMC gauge configuration generation jobs. The small white ovals and the large purple ovals represent XML input files and gauge field configuration files, respectively.

In this example, we propose putting all of the gauge field configuration files for a given parameter set into a single data set, seeded by the initial XML input file before the start of the session. This data set is represented by the dashed red rectangle in Figure 11. The teal dashed rectangle represents a separate data set to hold text output and log files (green triangles).

Listing 10.1 contains a workflow description file (WDF), describing the NFS services and namespaces used to capture the gauge files and other output. We have included a `datamover` (as described in Section 3.3) that will pre-fectch a needed gauge file checkpoint to the flash memory before a step start, and one that will archive a copy of all configuration files to tape upon the conclusion of the session.

Listing 10.2 lists the `iosea-ns` commands used to generate the two namespaces needed for step *A*, as well as the `iosea-wf` commands to execute the workflow. We stress that the code blocks in this section be considered pseudo-code, as the ephemeral services syntax is not finalized, and the LQCD application launch commands are simplified for readability.

```
workflow:
    name: step_A_HMC_wf
services:
    # namespace contains all gauge field files in ensemble defined by beta
```

```
    - name: nfs-gauge-files
      type: NFS
      attributes:
        namespace: "beta_{{ BETA }}_gauge_cfg_files"
        mountpoint:"/mnt/USER/gauge-fields/B{{ BETA}}/"
        flavor: medium
      datamovers:
      # promote a gauge file to flash before the start of a step
      - name: datamover1
        trigger: step_start
        target: flash
        operation: copy
        elements:
          - "*{{ ID }}*"
      # archive produced gauge files to tape at end of session
      - name: datamover2
        trigger: step_stop
        target: tape
        operation: move
        elements:
          - "*"
    # namespace for text output logs generated by HMC application
    - name: nfs-log-files
      type: NFS
      attributes:
        namespace: "step_A_logs_beta_{{ BETA }}"
        mountpoint:"/mnt/USER/text_logs/B{{ BETA }}/"
        flavor: medium

steps:
    - name: step_A_HMC
      location: gpu_module
      command: "sbatch {{ DEPENDENCY_FLAG }} chroma_HMC_beta{{ BETA }}.sh"
      services:
        - name: nfs-gauge-files RW
          datamovers:
            - datamover1
            - datamover2
       - name: nfs-log-files RW
```

Listing 10.1: `lqcd_A_workflow.yaml`, WDF for step *A*.

The text block in Listing 10.2 would not likely be run as a script, but rather interactively from a shell, and the step may have to be re-run multiple times as there is usually a policy limiting the number of jobs that a user may queue in the Slurm system. Note that the hypothetical shell script `get_most_recent_gauge_file_ID.sh` must contain an `iosea-wf access` command to be able to see the contents of the data set and its metadata.

```
beta=3.6

# create namespaces
iosea-ns create --auto-create-dataset beta_${beta}_gauge_cfg_files
iosea-ns create --auto-create-dataset step_A_logs_beta_${beta}

# fill namespace with initial step A input file
```

```
iosea-ns put beta_${beta}_gauge_cfg_files input_HMC_beta_${beta}_0000.xml

# start workflow
# session name: HMC_beta3.6
# need namespaces: beta_3.6_gauge_cfg_files step_A_logs_beta_3.6
iosea-wf start WORKFLOW=lqcd_A_workflow.yaml SESSION=HMC_beta${beta} BETA=$beta


# run the workflow
# session name: HMC_beta3.6
# step name: step_A_HMC
iosea-wf run SESSION=HMC_beta${beta} STEP=step_A_HMC ID=000

# check status
iosea-wf status SESSION=HMC_beta${beta}


# submit more if needed

# run some shell script to determine the most recent gauge file written
# Shell script must internally get interactive access to session!
num=‘get_most_recent_gauge_file-ID.sh /mnt/USER/gauge-fields/B${beta}/‘
iosea-wf run SESSION=HMC_beta${beta} STEP=step_A_HMC ID=$num


# stop the session and release the datanode
iosea-wf stop SESSION=HMC_beta${beta}

# stop should trigger datamover2 and a backup to tape archive
```

Listing 10.2: IO-SEA interface commands to launch workflow step *A*. Commands may be scripted or invoked interactively. A simple script determines the index of the most recent checkpoint gauge file in the directory and uses feeds this as an argument to the `iosea-wf run` command.Note that this must have an `iosea-wf access` command within to be able to access the data set with the gauge files.

```
> #start interactive session to check progress of gauge configuration files
> iosea-wf access SESSION=HMC_beta3.6
> ls -ltr /mnt/USER/gauge-fields/B3.6/ | tail
> less /mnt/USER/text_logs/B3.6/HMC_text_output_652.txt
```

Listing 10.3: IO-SEA interface commands to interactively check progress of workflow step *A*.

```
# lqcd-BC-workflow.yaml
workflow:
    - name: LQCD_BC

services:
    - name: nfs-gauge-files
      type: NFS
      attributes:
        namespace: "beta_{{ BETA }}_gauge_cfg_files"
```

```
        mountpoint:"/mnt/USER/gauge-fields/B{{ BETA }}/"
        flavor: medium

    - name: nfs_propagators
      type: NFS
      attributes:
        namespace: "propagators-b{{ BETA }}"
        mountpoint: "/mnt/USER/propagators/B{{ BETA }}/"
        flavor: medium

    - name: nfs_input-and-logs
      type: NFS
      attributes:
        namespace: "input-and-logs-b{{ BETA }}"
        mountpoint: "/mnt/USER/input-and-logs/B{{ BETA }}/"
        flavor: medium

    - name: nfs_hadron_correlators
      type: NFS
      attributes:
        namespace: "hadron-correlators-b{{ BETA }}"
        mountpoint: "/mnt/USER/corrs/B{{ BETA }}/"
        flavor: medium

steps:
    - name: "step_B_props-b{{ BETA }}"
          location: gpu_module # generate props on GPUs
      command: "sbatch {{ PROP_BATCH_SCRIPT }}"
      services:
        - "nfs-gauge-files" RW
        - "nfs_propagators" RW
        - "nfs_input-and-logs" RW
    - name: "step_C_contractions-b{{ BETA }}"
      location: cpu_module # hadron correlator contractions on CPUs
      command: "sbatch make_had_corrs-b{{ BETA }}.sh --export=id={{ ID }}"
      services:
        - "nfs_propagators" RO
        - "nfs_hadron_correlators" WO
        - "nfs_input-and-logs" WO
```

Listing 10.4: WDF for steps *B* and *C*: `lqcd_BC_workflow.yaml`.

As described above in Section 10.1, it is important in step *A* to have interactive access to the accumulating data while the workflow is running. It is normal to monitor the time between checkpoints, the Metropolis test acceptance rate, and to do simple diagnostic analysis on the data, checking e.g., the autocorrelation of the gauge field files to estimate the total number needed. In this case we would start interactive access to the `nfs-gauge-fields` service from a shell and then peer at the data as if it were a normal POSIX filesystem. An simple example of this is shown in the textblock in Listing 10.3.

```
#!/bin/bash
beta=3.6

#create namespaces for all hadron correlators:
iosea-ns create --auto-create-dataset hadron-correlators-b${beta}
```

```
#create namespace for input and log files for B & C
iosea-ns create --auto-create-dataset input-and-logs-b${beta}
# create empty namespace/dataset for propagators
iosea-ns create --auto-create-dataset propagators-b${beta}

start=500 # first 500 gauge files are not thermalized
skip=20 # we think it takes 20 updates for gauge fields to de-correlate
for (( num=start; num<=2000; num+=skip ))
do
    # also add input files for B and C to input-and-logs NS:
    iosea-ns put input-and-logs-b${beta} input_propsolve_beta_b${beta}_${num}.xml
    iosea-ns put input-and-logs-b${beta} input_had-corrs_beta_${beta}_${num}.xml
done

# start session
#session name: props-corrs-b3.6
iosea-wf start WORKFLOW=lqcd-BC-workflow-b${beta}.yaml SESSION=prop-corrs_b${beta} BETA=$beta

#run B step
for (( num=start; num<=2000; num+=skip ))
do
    iosea-wf run SESSION=prop-corrs_b${beta} STEP=step_B_props-b${beta} \
        PROP_BATCH_SCRIPT="solve_propagators-b${beta}-${num}.sh"
done

#check status of jobs:
iosea-wf status SESSION=prop-corrs_b${beta}

#run the C step
for (( num=500; num<=2000; num+=skip ))
do
    iosea-wf run SESSION=prop-corrs_b${beta} STEP=step_C_contractions-b${beta}} ID=$num
done

#check status of jobs
iosea-wf status SESSION=prop-corrs_b${beta}

#stop the session and release the datanode
iosea-wf stop SESSION=prop-corrs_b${beta}
```

Listing 10.5: IO-SEA interface commands to launch workflow steps *B* and *C*.

10.3.2. Steps B & C: Computing propagators and hadron correlators

LQCD workflow steps *B* and *C* are tightly coupled and are sometimes combined in common compute jobs. In this example we treat them as separate steps. Recall that in step *B* a linear solver produces propagators using gauge field file as input. We save the propagators to disk in this example. In step *C* several of the propagators are read in and contracted together to form hadron correlator arrays.

We propose introducing a new data set to hold all of the propagators produced in Step *B*. This is represented by the blue dashed rectangle in Figure 11. A separate data set (green dashed rectangle in Figure 11) will hold *all* of the produced hadron correlators, which must be analysed together in

step *D*. As with step *A* a data set will hold text output logs, and in this case input files (represented in Figure 11 by the same teal dashed rectangle).

The code block in Listing 10.4 has a YAML WDF describing the ephemeral services for this step *BC* scheme. Note that this file uses the proposed parametric functionality and can be used as a WDF file for an entirely different ensemble and workflow parameterized by a different `${beta}` value.

The code block in Listing 10.5 shows a sample of the commands invoked to create the step *B* and *C* data sets and run the steps for the case of `beta=3.6`, iterating over the index `num`.

10.3.3. Step D: Offline analysis

In step *D* we analyse the collection of hadron correlators generated in step *C*. We consider the case of interactively running analysis scripts. A WDF file for providing access to the data set containing all of the hadron correlators is given in Listing 10.6.

The sample shell script in Listing 10.7 activates interactive access to the data set with namespace `hadron-correlators-b3.6`

```
#lqcd_D_workflow.yaml
workflow:
    - name: "LQCD_D_B{{ BETA }}"

services:
    - name: "nfs_hadron_correlators-b{{ BETA }}"
      type: NFS
      attributes:
        namespace: hadron-correlators-b{{ BETA }}"
        mountpoint: "/mnt/USER/corrs/B{{ BETA }}/"
        flavor: medium
```

Listing 10.6: WDF for step *D*, `lqcd_D_workflow.yaml`. Note that no `step` is defined, as analysis is through an interactive session.

```
#!/bin/bash
beta=3.6

#start a workflow session named "analyse_b${beta}_corrs"
iosea-wf start WORKFLOW=lqcd_D_workflow.yaml SESSION=analyse_b${beta}_corrs BETA=$beta

# get interactive access to dataset with hadronic correlators
iosea-wf access SESSION=analyse_b${beta}_corrs

# run analysis scripts on directory with hadron correlators
amazing_result_from_hadcorrs.pl /mnt/USER/corrs/B${beta}/hadron-correlators-b${beta}

iosea-wf stop SESSION=analyse_b${beta}_corrs
```

Listing 10.7: Simple, hypothetical analysis script using interactive access to a workflow.

10.3.4. Putting it all together

For readability, in the subsections above we have put the different workflow steps into separate workflow sessions, except for the tightly coupled steps *B* and *C*. There is a tidy logic to this scheme. However, it may not be possible for two separate workflows to have synchronised access to the same data set, even if one session only requires read-only access. That would mean it is not possible to start step `B` jobs calculating propagators while step *A* continues to produce gauge configuration files. So combining the four steps in a single workflow session may be the favored solution.

On the other hand, having a single combined session seems to imply that one needs to know all possible analysis steps one might wish to launch, and code them in the WDF, before launching step *A*.

While the optimal scheme will be evident only after experience on a prototype system, we can imagine a solution where we start step *A* with a WDF like in Listing 10.1. When enough gauge configuration files have accumulated to start step *B* and *C*, we stop the session at a convenient point (waiting for queued jobs to finish), add the needed services and steps (as listed in Listing 10.4) to the original WDF, then restart the session and run steps *A*, *B*, and *C*. We would iterate this process, adding further steps to the WDF, as the project dictated the need for various calculations or data analysis.

10.4. LQCD Summary

We see in the preceding sections a candidate scheme to integrate the LQCD workflow into the IO-SEA environment. We have tried to explore what appeared to be useful features of the IO-SEA workflow manager, such as parametric WDF functionality.

In the course of developing this example we explored several different candidate schemes. We discovered a few might not be functional, but for the feasible schemes it is difficult to fully identify the advantages and disadvantages each presents. We expect that our idea of the “optimal” scheme will evolve significantly when we are able to test a full-scale LQCD simulation on prototype machine running IO-SEA services. In fact, it is likely that in real tests we may discover issues with LQCD workflow schemes that are not currently apparent.

However, we think after eventual hands-on experience it will be possible to develop several workable LQCD template examples that LQCD practitioners can apply to their varied scientific projects.

Part IV.

Summary

11. Summary

This deliverable report documents the plans to integrate the scientific use cases into the IO-SEA environment. These plans are the result of many detailed discussions between the use cases and the other technical work packages of the IO-SEA project, in particular Work Package 2 who focus on the design of the ephemeral services interface.

These discussions, and the process of understanding which services each application would use, and how they would use them, has naturally led to refinement of the design of the IO-SEA environment. We have tried to document this productive co-design process in Part I, and to provide a complete description of the updated ephemeral services and workflow management interface in Part II, a part of the report which has already proven to be a useful internal reference for the users when describing the integration of their use cases in Part III.

We do not expect that the designs laid out in this report will remain unchanged. We feel, however, that the process of developing initial, detailed use-case workflow schemes for the IO-SEA environment has left the use cases very well-positioned for the forthcoming task of porting the use-case applications to prototype machines running IO-SEA services.

List of Acronyms and Abbreviations

A

AMR	Adaptive Mesh Refinement is a grid management technique that allows for adaptive resolution to help reach a better compromise between memory consumption, numerical precision and computational speed (see [9]).
API	An Application Programming Interfaces (API) allows software to communicate with other software which support the same API.
ASCII	American Standard Code for Information Interchange.
ATOS	ATOS is one of Europe's largest digital-services deliverers.

B

BB-NFS	Burst-buffered Network File System, with a fast intermediate storage layer positioned between the front-end computing processes and the back-end storage systems.
BeeGFS	BeeGFS is a parallel file system, developed and optimized for high-performance computing.

C

CEITEC	Central European Institute of Technology, Masaryk University, Brno, Czech Republic.
CFD	Computational fluid dynamics.
CHROMA	The Chroma software system for lattice QCD.
CLI	Command-line interface.
CLM	Community Land Model.
CORDEX	Coordinated Regional Downscaling Experiment. A framework to evaluate regional climate model performance through a set of experiments aiming at producing regional climate projections. It includes standard domain definitions of interest for TSMP.
CORTX	CORTX is hardware-agnostic open-source object storage software designed by SEAGATE for mass capacity-optimized data storage architectures.
COSMO	Consortium for Small-scale Modeling.
CPU	Central Processing Unit.

D

DASI Data Access and Storage Interface developed in Work Package 5.

E

ECMWF European Centre for Medium-Range Weather Forecasts.

EMPIAR The Electron Microscopy Public Image Archive, is a public resource for raw images underpinning 3D cryo-EM maps and tomograms.

F

FPGA Field-Programmable Gate Array.

FZJ Forschungszentrum Jülich, in Jülich, Germany, is one of the largest research centres in Europe and a member of the Helmholtz Association.

G

GBF Global Bunch of Flash, a storage services with no connection to the long term storage. GBF is for multiple compute nodes shared temporary POSIX file system, in contrast to SBF.

GPU Graphics Processing Unit.

H

HDD Hard Drive Disk.

HDep Hercule “Depouillement”, post-processing database.

Hercule Parallel I/O and data management library developed at CEA. Library evolutions are part of WP5.

HMC Hybrid Monte Carlos (or Hamiltonian Monte Carlo) is a Markov chain Monte Carlo method for obtaining a sequence of random samples. Hamiltonian dynamics are used to propose an update to the random variables, and the update is accepted or rejected by a Metropolis test.

HPC High-Performance Computing.

HProt Hercule “Protection”, Checkpoint/restart database.

HSM Hierarchical Storage Management.

I

I/O Input/Output is either a noun referring to the action of doing either input and/or output, generally either reading or writing memory, or is an adjective or adverb that describes that the following operation does input and/or output.

IFS Integrated Forecasting System, ECMWF’s operational weather forecasting system.

iRODS Integrated Rule-Oriented Data System.

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

K

Kronos The ECMWF workload simulator.

L

LQCD Lattice quantum-chromodynamics is a numerical framework for calculating physical properties of hadrons, composite particles composed of quarks.

M

MARS The Meteorological Archival and Retrieval System is ECMWF's perpetual archive service.

MPI The Message Passing Interface is a common API for communication between tasks running on one or more computers.

MSA Modular Supercomputing Architecture.

MU Masaryk University, Brno, Czech Republic.

N

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.

NVRAM Non-volatile RAM is RAM that does not lose the stored information after a short time without constant refreshing.

O

OASIS OASIS3-MCT is a software allowing synchronized exchanges of coupling information between numerical codes representing different components of the Earth System.

P

ParFlow A physically-based and spatially distributed hydrological model solving surface and subsurface flows in a massively parallel computational framework.

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

Q

QIO QCD Input/Output Applications Programmer Interface developed under the auspices of the U.S. Department of Energy Scientific Discovery through Advanced Computing (SciDAC) program.

R

RAID A “redundant array of independent disks” is a data storage virtualization technology that combines multiple physical disk drive components into one or more logical units for the purposes of data redundancy, performance improvement, or both.

RAM Random Access Memory is memory optimised for random access, often by a compute device.

RAMSES Code to model astrophysical systems, featuring self-gravitating, magnetised, compressible, radiative fluid flow, using AMR technique. French acronym for "Effortless adaptive mesh refinement".

RO Abbreviation of Read Only.

RW Abbreviation of Read Write.

S

S3 Amazon’s Simple Storage Service: HTTP-based protocol to access data. Initially developed by Amazon, it generalisation made it a de-facto standard for data access in cloud services.

SBF Single Bunch of Flash, a temporary flash storage services with no connection to the long term storage. SBF is for a single compute node using the temporary POSIX file system, in contrast to GBF.

SEAGATE Seagate is one of the largest data storage providers.

Slurm Slurm is an open-source cluster-management and job-scheduling system.

SSD Solid State Drive.

T

TIFF The Tag Image File Format is an image file format.

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.

WO Abbreviation of Write Only.

X

XFS XFS is a high-performance 64-bit journaling file system.

XML XML, extensible markup language is a markup language and file format for storing, transmitting, and reconstructing arbitrary data.

Y

YAML YAML is a human-readable data-serialization language.

Bibliography

[1] E. Borba, P. Couveé, W. Frings, F Guimarães, S. Krempel, S. Mimouni, V Silva, D Vasiliaukas, and I. Zhukov. IO-SEA D3.1: Instrumentation and monitoring, concepts and architecture. Technical report, IO-Software for Exascale Architectures, Apr. 2022.

[2] T Leibovici, S. Gougeaud, P. Lucas, G. Courrier, P. Deniel, Kannan V. Gursoy B., G. Umanesan, D. Vasiliauskas, P. Couvée, S. Derr, and Raaf P. IO-SEA D4.1: Hierarchical storage management features, concepts and architecture. Technical report, IO-Software for Exascale Architectures, Apr. 2022.

[3] J. Hawkes and O. Iffrig. IO-SEA D5.1: First version of the Data Access and Storage Interface (DASI) implementation. Technical report, IO-Software for Exascale Architectures, Apr. 2022.

[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, Feb 2022.

[5] E. B. Gregory, P. Couvée, and M. Golasowski. IO-SEA D1.1 Application and co-design input. Technical report, IO-Software for Exascale Architectures, jul 2021.

[6] DEEP-SEA Project. https://www.deep-projects.eu/.

[7] 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.

[8] Robert G. Edwards and Balint Joo. The Chroma software system for lattice qcd. *Nucl. Phys. B Proc. Suppl.*, 140:832, 2005.

[9] A. Khokhlov. Fully Threaded Tree Algorithms for Adaptive Refinement Fluid Dynamics Simulations. *Journal of Computational Physics*, 143(2):519–543, July 1998.