Progress of WP4: Data at Scale

WP4 Team

ESiWACE GA

27 May 2020

CENTRE OF EXCELLENCE IN SIMULATION OF WEATHER AND CLIMATE IN EUROPE

Reminder: WP4: Data Systems at Scale

Objectives

To mitigate the effects of the data deluge from high-resolution simulations (project objective d) by

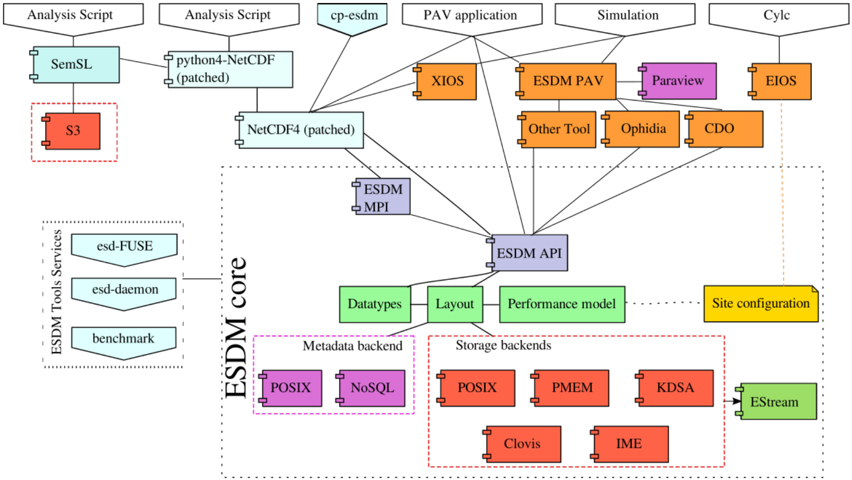
1. Supporting **data reduction in ensembles** by providing tools to carry out ensemble statistics "in-flight" and compress ensemble members
2. **Hiding complexity** of multiple-storage tiers (middleware between NetCDF and storage) with industrial prototype backends
3. Delivering **portable workflow support** for manual migration of semantically important content between disk, tape, and object stores

$\Rightarrow$ *Ensemble tools, storage middleware, storage workflow*

Outline

Design and Leadership: Architecture/Interactions

- Created a design that indicates the interactions between relevant software



Outline

T2: Ensemble Services

Reminder: Goals

- Run coupled ensemble members that via XIOS create less data
  - e.g., store mean and variance of ensemble results (instead of all members)

Ongoing activities

- Implemenation of UM-XIOS output on reduced gaussian grid
  - Ensemble of 10km UMs w/reduced gaussian
- Further performance analysis with time-processed ensemble output
- Investigations with second-level XIOS servers and compression
- Developing/Evaluating an XIOS-ESDM Cylc test framework

Outline

Reminder: Earth-System Data Middleware (ESDM)

A transitional approach towards a vision for I/O addressing

- Scalable data management practice
- The inhomogeneous storage stack
- Suboptimal performance and performance portability
- Data conversion/merging

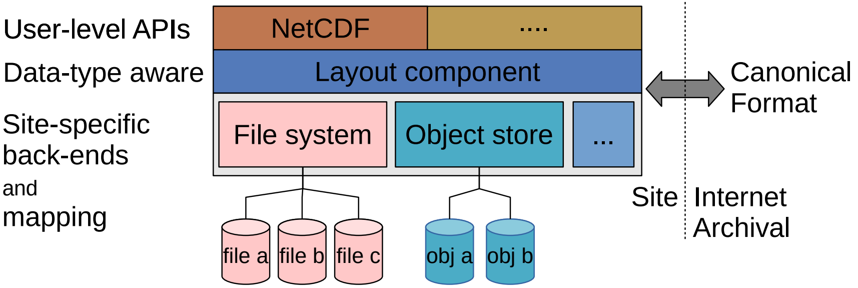
Design goals of the Earth-System Data Middleware

1. Relaxed access semantics, tailored to scientific data generation
2. Site-specific (optimized) data layout schemes
3. Ease of use and deploy a particular configuration
4. (Enable a configurable namespace based on scientific metadata)

Reminder: Architecture

Key concept: Decouple data localization decisions from science

- Middleware utilizes layout component to make placement decisions
- Applications work through existing API
- Data is then written/read efficiently; potential for optimization inside library



Selected Activities: Status Overview

- Usability testing with relevant applications (works/minor issues to resolve)
  - Ophidia, CDO (using ESDM/NetCDF)
  - Dask (reading/writing ESDM/NetCDF)
  - XIOS (using ESDM/NetCDF)
- Implemented ESDM as API in a shallow water model to show all features
  - Will be used for demonstrating post-processing (WP5) too
- Hardening (bug fixes, documentation, reorganization, maintainability)
- Optimization (read path, fragment handling, non-consecutive/data holes, FORTRAN handling)
- Created streaming API to minimize memory pressure
- Support compression in ESDM using SCIL (decouples accuracy from decision)
- Support data replication upon read to optimize placement (evaluation pending)
- Build prototypes for supporting post-processing, analytics and (in-situ) visualization

ESDM as NetCDF Drop-In is Easy to Use

- Create a ESDM configuration with storage locations
- Run esdm-mkfs to prepare storage systems (e.g., mkdir on POSIX)
- Change file names when running NetCDF applications
  - The namespace of ESDM is separated from the file system (hierarchical too)
  - NetCDF can use ESDM by just utilizing the **esdm://** prefix
- Examples:
  - Import/Inspection/Export of data using NetCDF
    $ nccopy test_echam_spectral.nc esdm://user/test_echam_spectral
    $ ncdump -h esdm://user/test_echam_spectral
    $ nccopy -4 esdm://user/test_echam_spectral out.nc
  - Usage in XIOS, change iodef. Example:
    `<file id="output" name="esdm://output" enabled=".TRUE.">`
    prec=8 in axis_definition, domain_definition and field_definition

Converting an Existing Code: Shallow Water Model

Facts about the model

- Stores data column-wise in memory
- Separates compute phase and IO phase[1]

Existing NetCDF code for IO phase

```
size_t start[] = {0, 0};
size_t count[] = {nY, 1};
for(unsigned int col = 0; col < nX; col++) {
  start[1] = col; //select col (dim "x")
  nc_put_vara_float(dataFile, i_ncVariable, start, count,
    &i_matrix[col+boundarySize[0]][boundarySize[2]]);
}
```

[1] DSLs will help to separate those phases

ESDM Code for the Application

```
int64_t offset[] = {(int64_t) timeStep, offsetY, offsetX};
int64_t size[] = {1, (int64_t) nY, (int64_t) nX};

esdm_wstream_float_t stream;
esdm_wstream_start(&stream, dset, 3, offset, size);
for(int y = 0; y < nY; y++) {
 for(int x = 0; x < nX; x++) {
   esdm_wstream_pack(stream,
       i_matrix[x + boundarySize[0]][boundarySize[2] + y])
   // this may trigger actual IO and postprocessing!
 }
}
esdm_wstream_commit(stream);
```

- Ultimately, using DSLs an IO phase could mix in compute and "stream output" to minimize memory pressure (and trigger initial post-processing)

Supported Backends

Storage backends

- POSIX: Backwards compatible for any shared storage
- CLOVIS: Seagate-specific interface, will be open sourced soon
- WOS: DDN-specific interface for object storage
- KDSA: Specific interface for the Kove cluster-wide memory
- **PMEM (NEW):** Non-volatile storage interface (`http://pmem.io`)
- **IME (NEW):** DDN's Infinite Memory Engine

Metadata backends

- POSIX: Backwards compatible for any shared storage
- Investigating ElasticSearch, MongoDB as potential NoSQL solutions

Performance on NVDIMMs

- ESDM on the NextGenIO Prototype with a first naive approach (with PMEM)
  - ▶ Test run on four dual-socket nodes with 80 GByte of data
  - ▶ Theoretic HW performance per node (12 NVDIMMs) W: 96 GB/s, R: 36 GB/s
- Compare POSIX Optane vs. using NVDIMM Optane (ESDM PMEM backend)
  - ▶ Similar to TMPFS performance in read path
- Max test: explore potential best case performance (single file)
- Optimizations are possible (ported backend was a quick hack)

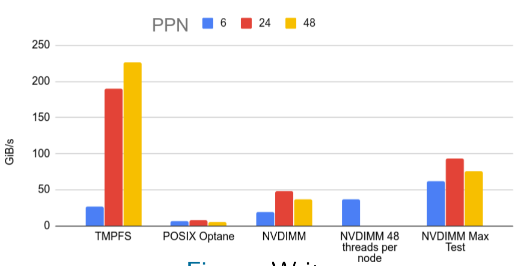
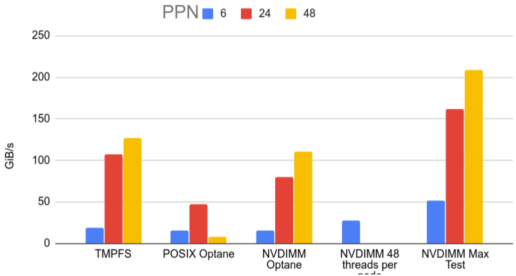


Figure: Write



ESiWACE2 TODOs for ESDM

- Hardening and optimization of ESDM
  - Integrate improved performance model
  - Backend optimization
- Features
  - Complete replicate data upon read (adaptive fragments)
  - NoSQL metadata backend
  - S3 backend
- Evaluation of structured (chunked) vs. flexible (ESDM) fragments
- Evaluation of ESiWACE-relevant scenarios
- Industry proof of concepts for EDSM, i.e., shipping of HW with software
- Supporting post-processing, analytics and (in-situ) visualization
  - Support of computation offloading within ESDM
  - Evaluation using analysis tools, e.g., Ophidia, CDO

Outline

JDMA: Joint Data Migration App

Reminder: Joint Data Migration App

- Aims to manage large data migrations on behalf of a user
  - Keeping record of manifest, carrying out checksums, and recording state

Status

- In production use on JASMIN (to both tape and object store)
  - Over 700TB transferred and catalogued from the RDF on Archer
- Users positive about functionality, but not performance, particularly to tape
- Performance (in particular, throughput) is not yet meeting expectations
  - Largely due to the verification process - pulled from tape and checksums compared

Next step

- Considering how experience thus far can be used to inform refactoring

Reminder: S3NetCDF (Python Module)

Drop in replacement for NetCDF which understands S3 object stores

- Utilises the CFA data model to aggregate objects; each is a valid netCDF file

CFA and S3NetCDF

- Climate and forecast (CF) aggregation rules describe how multiple CF fields may be combined into one larger field
  - A **master array** file (kBs in size): Domains and metadata for a number of variables; Coordinates for the domains; Metadata for the subarrays, position in the master array; No field data
  - A number of **subarray** files (MBs to GBs in size): Subdomain and metadata (replicated from master array); Coordinates for the subdomain; **Field** data
- Redundant information: master for efficiency, replication in subarrays for reliability

S3NetCDF

News 2019/20

- Complete rewrite of all code!
- New master file format for CFA (v0.5, now uses NetCDF4 groups)
- Pluggable frontend parsers to exploit CF — currently netCDF3/4
  - Planning for ESA SAFE, GRIB, PP
- Pluggable backends: written as Python file objects with seek, tell, read, write, ...
  - Supports two S3 versions: vanilla and asyncio
- Now completely cacheless; read/write direct to disk/memory/S3
- New aggregation tool (creates master array file from existing files)
- New information tool (like ncdump, shows info about master/subarray files)
- Unit-tests, interface consistency tests, continuous integration

S3NetCDF

Status

- Nearly feature complete v2 in Github (fully useable, but only supports uniform partitions and doesn't include memory management)
- `s3nc_cda_info` tool feature complete; `s3nc_cfa_agg` aggregation tool usable (but only 1d aggregation)

Next steps

- Working on feature-completeness, documentation, tutorials (using CMIP6 on CEDA Caringo via S3)
- Release of v2 and publish accompanying paper this year

Outline

Reminder: T5: Workflows

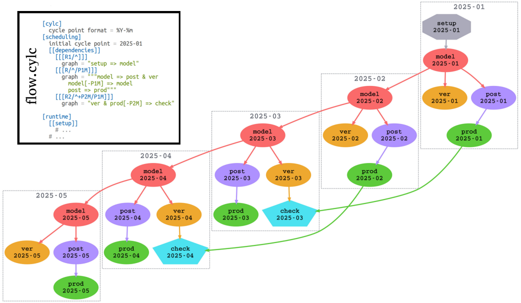
- Goal: Explore higher-level abstraction - scientists don't need to worry where data is
- Data placement could be optimized by considering available hardware
  - Different and heterogenous storage systems available
  - Prefetching of data, using local storage, using IME hints, ...
- Status: We created a design document in the consortium

- A workflow consists of many steps
  - Repeated for simulation time
  - E.g., weather for 14 days
- Cylc workflow specifies
  - Tasks with commands
  - Environment variables
  - Dependencies

flow.cylc

```
[cylc]
  cycle point format = %Y-%m
[scheduling]
  initial cycle point = 2025-01
  [[dependencies]]
    [[[R1/^]]]
      graph = "setup => model"
    [[[R/^/P1M]]]
      graph = """model => post & ver
        model[-P1M] => model
        post => prod"""
    [[[R2/^+P2M/P1M]]]
      graph = "ver & prod[-P2M] => check"

[runtime]
  [[setup]]
    # ...
  # ...
```



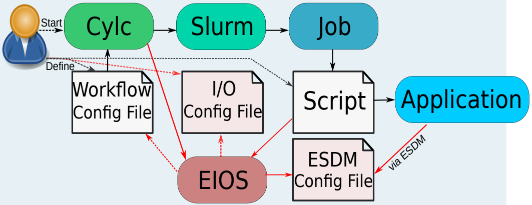
Design Overview for Workflow Extensions

Relevant components

- Configuring system information
- Extending the workflow description (with IO inputs needed and output specification)
- Providing a smart I/O scheduler (EIOS)

Modified workflow execution

1. Cylc analyzes workflow
   - EIOS provides Slurm variables
2. Wflow manager allocates resources
   - May schedule on nodes of prev. jobs
3. Job script runs applications
   - EIOS generates pseudo filenames encoding scheduling information



Smarter I/O Scheduler: Benefits

- Abstraction: Decouple decision making about storage location(s) from scientists
- Scheduler will provides hints for colocating tasks (application runs) with data
  - Create dummy file name to include schedule (e.g., prefer local storage)
  - ESDM parses the schedule information and enacts it (if possible)
- Optimizing data placement strategy in ESDM/workflow scheduler will be applied
  - Utilizing hints for IME to pin data to cache
  - Storing data locally between depending tasks (using modified Slurm)
  - Optimizing initial data allocation (e.g., alternating storage between cycles)

Outline

Active Storage/Compute offloading/Server-Side Computation

Highlight: Active Storage

- Activity supports WP5 activities
- Collaboration with DDN (and Seagate)

Approach

- Send compute function for reduction to storage servers, e.g., min/max
- Server runs the function on the data and replies with data
- Client will only need to merge the results
- Useful for CDO and Ophidia

Benefit: A Simple Performance Model

Assumptions

- Need to compute min/max for 1000 GB of data, e.g., with CDO
- IME storage system provides 1000 GB/s, client performance: 12.5 GB/s

Traditional approach

- 1 Client Node: 80s just to read data
- 80+ Client Nodes to saturate network/IME: 1s to read data

With active storage

- 1 Client process submits reduction to servers, IME processes with 1000 GB/s
- Servers return << 1 GB of data
- Total runtime: 1s, thus need less clients to achieve same performance

Conclusions and Discussion

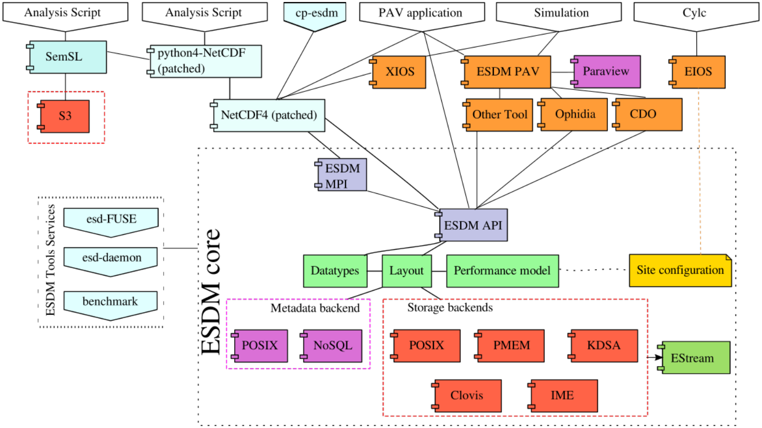
Coordination

- Our goal is to use the tools in relevant workflows

⇒ Talk with WP1 to explore usage of IO stack/tools with hi-res workflows

- Is there a way to create a small demonstrator that can be open sourced that showcases ESiWACE tools?

Architecture: Detailed View of the Software Landscape



Data Model

- Container:
  - Provides a flat (simple hierarchical) namespace
  - Contains Datasets + (arbitrary) metadata
  - Can be constructed on the fly
- Dataset:
  - Multi-dimensional data of a specified data type
  - Write-once semantics (epochs are planned)
  - Contains arbitrary number of data fragments
  - Data of **different fragments** can be **disjoint or overlapping**
  - Dimensions can be named and unlimited
  - Self-describing, can be linked to multiple containers
- Fragment:
  - Holds data, arbitrary continuous sub-domain (data space)
  - Stored on exactly one storage backend

Discussion of the Data Model

1. Fragment domain is flexible
   - Avoid false sharing (of data blocks) in the write path
   - A fragment can be globally available or just locally
   - Reduce penalties of **shared** file access
2. Self-describing data format
   - Metadata contains relevant scientific metadata, datatypes
3. Layout of the fragments can be dynamically chosen
   - Based on site-configuration and performance model
   - Site-admin/project group defines a mapping
   - Use multiple storages concurrently, use local storage
4. Containers could be created on the fly to mix-in datasets
   - Open one container for input that has everything you need

Metadata of a Complex File: The NetCDF Metadata

```
netcdf test_echam_spectral {
dimensions:
        time = UNLIMITED ; // (8 currently)
        lat = 96 ;
        lon = 192 ;
        mlev = 47 ;
        ilev = 48 ;
        spc = 2080 ;
        complex = 2 ;
variables:
        float abso4(time, lat, lon) ;
                abso4:long_name = "antropogenic sulfur burden" ;
                abso4:units = "kg/m**2" ;
                abso4:code = 235 ;
                abso4:table = 128 ;
                abso4:grid_type = "gaussian" ;
        ... [126+ more variables] ...
// global attributes:
                :CDI = "Climate Data Interface version 1.4.6
                    ↪ (http://code.zmaw.de/projects/cdi)" ;
                :Conventions = "CF-1.0" ;
                :source = "ECHAM6.1" ;
                :institution = "Max-Planck-Institute for Meteorology" ;
                ... 10 more attributes ...
                :NCO = "4.4.5" ;
}
```

Mapping by the POSIX Metadata Storage

Stored metadata inside the metadata directory

```
containers/user/test_echam_spectral.nc.md
datasets/VZ/zMKbbzj9Y0kEpk.md
        ... for each dataset one file ...
```

Metadata is stored as JSON: the container

```
{
  "Variables": { # Metadata of the global attributes
      "childs": {
      "CDI": {
              "data": "Climate Data Interface version 1.4.6
                  ↪ (http://code.zmaw.de/projects/cdi)"
              "type": "q71@l" # The datatype ASCII encoded
          },
       },
  }
  "dsets": [
      {
          "id": "VZzMKbbzj9Y0kEpk",
          "name": "abso4"
```

Metadata is stored as JSON: a dataset

```
{ "Variables": {
    "childs": { # Attributes...
    "grid_type": { "data": "gaussian", "type": "q8@l"}
} },
"dims": 3, # dimensionality of the data
"dims_dset_id": [ "time", "lat", "lon"], # the named dimensions
"fill_value": {"data": 9.96920997e+36, "type": "j"},
"size": [0, 96, 192], # the dimensionality of the data, here unlimited 1st dim
"typ": "j" # The type of the data, here float
"id": "VZzMKbbzj9Y0kEpk", # ID of the dataset
"fragments": [
  {"id":"VZzMKbGtnusZsRVv3Pky","pid":"p1","size":[1,96,192],"offset":[0,0,0]},
  {"id":"VZzMKbRhYpl6cOl0frBX","pid":"p1","size":[1,96,192],"offset":[1,0,0]},
  ...
  {"id":"VZzMKbl8JyXk4fUXfwrS","pid":"p1","size":[1,96,192],"offset":[7,0,0]}]
}
```

Mapping of Fragments by Storage Backends

Mapping of the POSIX storage

- A fragment is mapped into a file: `<dataset>/<fragmentID>`
- Contains the raw data
- Optionally suffixed by some metadata to allow "restoration" of broken storage

Mapping of the KDSA storage

- Volume of shared memory is partitioned into blocks
- Block header describes free/occupied blocks
- Atomic operations to aquire/free a block
- A block stores one fragment; ID is the offset into the volume