Data Systems at Scale in Climate and Weather: Activities in the ESiWACE Project

Julian Kunkel on behalf of the ESiWACE WP4 Team

Department of Computer Science, University of Reading

25 June 2020

esiwace
CENTRE OF EXCELLENCE IN SIMULATION OF WEATHER AND CLIMATE IN EUROPE

Outline

Disclaimer: This material reflects only the author's view and the EU-Commission is not responsible for any use that may be made of the information it contains

ESiWACE: http://esiwace.eu

The Centre of Excellence in Simulation of Weather and Climate in Europe

- Prepare the European weather and climate community
  - Make use of future exascale systems
- Goals in respect to HPC environments
  - Improve efficiency and productivity
  - Supporting the end-to-end workflow of global Earth system modelling
  - Establish demonstrator simulations that run at the highest affordable resolution
- Funding via the European Union's Horizon 2020 program (ESiWACE2 2019-2022)

esiwace
CENTRE OF EXCELLENCE IN SIMULATION OF WEATHER AND CLIMATE IN EUROPE

The ESiWACE Community

- 20 partners from 9 countries
- 35 supporters

Figure: Group Photo during the ESiWACE2 Kick-Off Meeting (March 2019)

Climate/Weather Workflows

Challenges

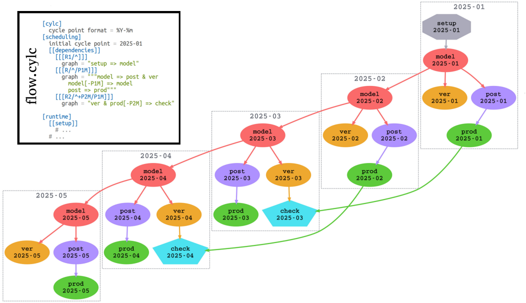
1. Programming of efficient workflows
2. Efficient analysis of data
3. Organizing data sets
4. Ensuring reproducibility of workflows/provenance of data
5. Meeting the compute/storage needs in future complex hardware landscape

Scientists should rather focus on 1 and 2

Workflows in Climate and Weather

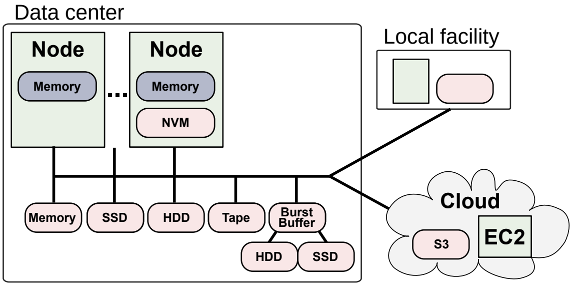
esiwace
CENTRE OF EXCELLENCE IN SIMULATION OF WEATHER AND CLIMATE IN EUROPE

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



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

The Coexistence of Storage – Impact of Local Storage



- May utilize local storage, SSDs, NVMe
  - Even without communication used in workflows
- Goal: We shall be able to use all storage technologies concurrently
  - Without explicit migration, put data where it fits
  - Administrators just add new technology (e.g., SSD pool) and users benefit from it

Outline

Long Term Vision: Full Separation of Concerns

Decisions made by users/scientists

- Scientific metadata (e.g., what is the data about)
- Declaring workflows
  - Covering data ingestion, processing, product generation, and analysis
  - Data life cycle (and archive/exchange file format)
  - Declaring value of data (logfile, data-product, observation)
  - Constraints on: accessibility (permissions), ...
  - Expectations: completion time (interactive feedback human/system)
- Flexibly adapt to needs of users/scientists
  - Modify workflows on the fly
  - Analyse interactive, e.g., Visual Analytics

Separation of Concerns

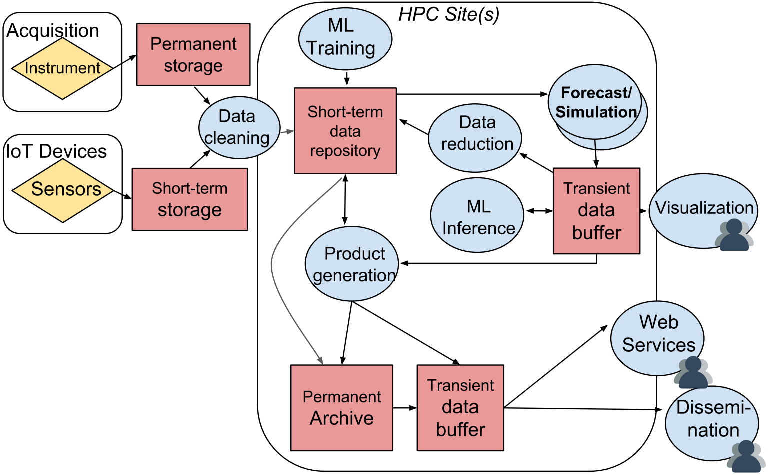
Decision made by programmers of models/tools

- Decide about the most appropriate API to use (e.g., NetCDF + X)
- Register compute snippets (analytics) to API
- Do not care **where** and **how** compute/store

Decisions made by the (compute/storage) system

- Where and how to store data, including file format
- Complete management of available storage space
- Performed data transformations, replication factors, storage to use
- Including scheduling of compute/storage/analysis jobs (using, e.g., ML)
- Where to run certain data-driven computations (**Fluid-computing**)
  - Client, server, in-network, cloud, your connected laptop

Smarter Climate/Weather Workflows in 2020+



- IoT (and mobile devices)
  - Additional data provider
  - Improves short-term weather prediction
- Machine learning support
  - Localize known patterns
  - Interactive use
    Visual analytics
- Data reduction
  - Output is triggered by events (ML)
  - Compress data of ensembles

Outline

Earth-System Data Middleware

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

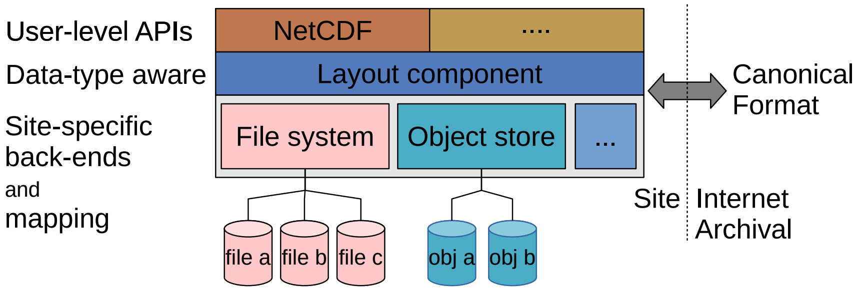
Design goals of the Earth-System Data Middleware

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

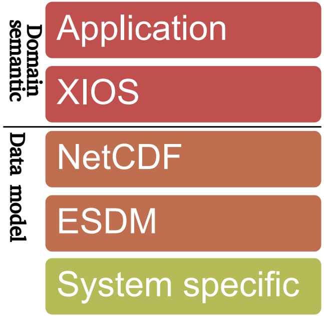
Architecture

Key concepts

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

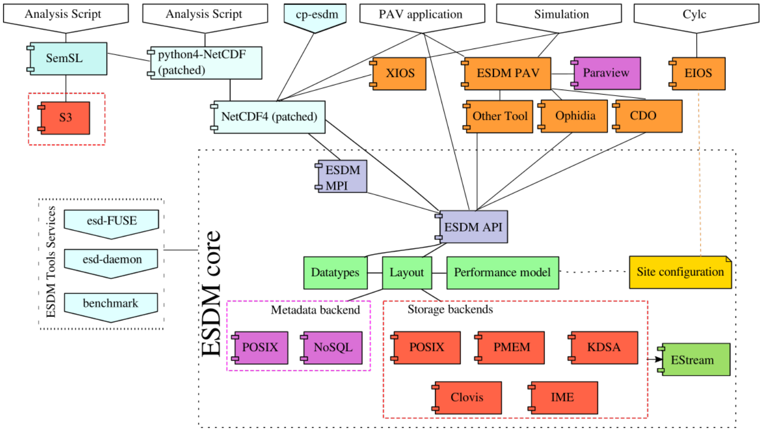


A Transitional Storage Stack for Large-Scale Ensembles



- Users run ensemble (e.g., 10x simulation with slightly different parameters)
- XIOS (climate/weather domain-specific) servers run on subset of nodes
  - Receive data from all 10 simulations
  - Reduces data, e.g., computing mean/variance
  - Store interesting data (reduced data and maximum)
- ESDM performs IO efficiently
  - Using underlying (heterogenous) storage systems efficiently

Architecture: Detailed View of the Software Landscape in ESiWACE



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: Non-volatile storage interface (`http://pmem.io`)

Metadata backends

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

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)

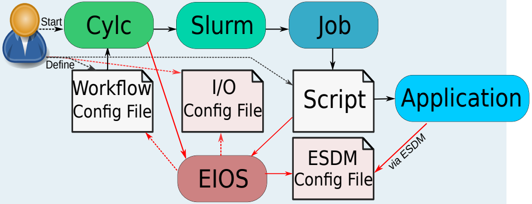
Design Overview for Workflow Extensions

Relevant components

- Configuring system information
- Extending the workflow description (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

Evaluation

System

- Test system: DKRZ Mistral supercomputer
- Nodes: 100, 200, 500

Benchmark

- Uses ESDM interface directly; metadata on Lustre
- Write/read a timeseries of a 2D variable; 3x repeated
- Grid size: 200k $\times$ 200k $\times$ 8 Bytes $\times$ 10 iterations
- Data volume: size = 2980 GiB; compared to IOR performance

ESDM configurations

- Splitting data into fragments of 100 MiB
- Use /dev/shm (TMPFS) or /tmp directory (Local SSD)

Performance Growth of ESDM on Lustre (PPN = 1)

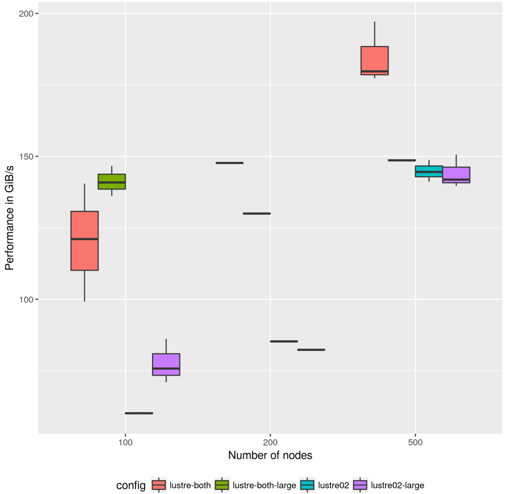


Figure: Write

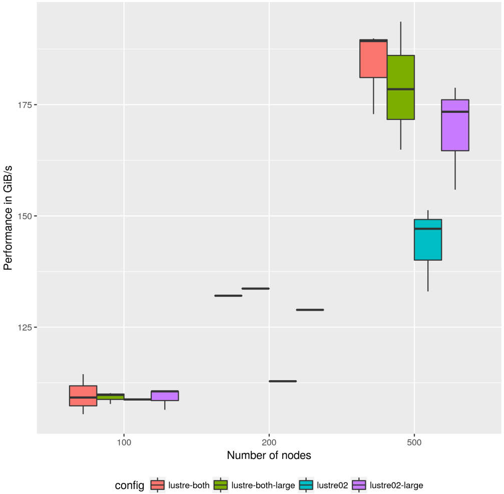


Figure: Read

Discussion

- Benefit when accessing multiple global file systems
- Write performance benefits from using both file systems
  - Most benefit when using 200 nodes (2x)
  - 500 nodes: 180 GiB/s vs. 140 GiB/s (single fs)
- Read performance shows some benefit for larger configurations
- ESDM achieves similar performance regardless of PPN (not shown)
- What is the performance when we use node-local storage?

Performance on TMPFS vs. IOR (nodes = 500, varied PPN)

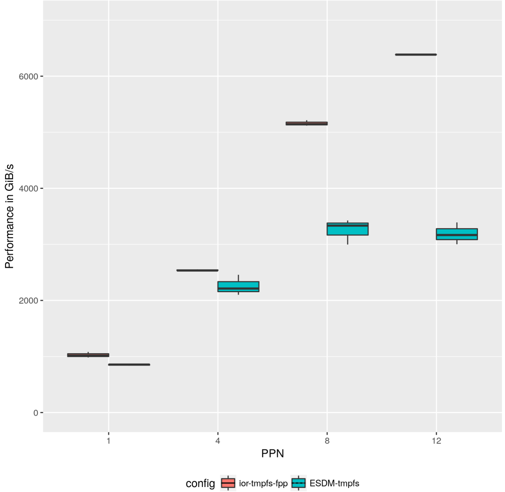


Figure: Write

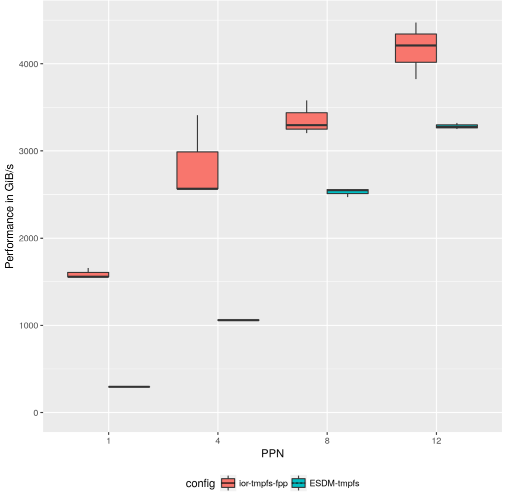


Figure: Read

Discussion

- Node-local storage is much faster than global storage
  - TMP achieves 750-1,000 GB/s for write (500 SSDs, some caching)
  - TMP reads are actually cached (6 GB data per node)
  - TMPFS achieves up to 3,000 GB/s
- TMP write is invariant to PPN
  - ESDM configured to use at least four threads per node
- TMPFS write depends on PPN
  - ESDM configured to not use threads, could use them to improve performance!
- IOR is faster; potential to improve ESDM path further
  - Localization of fragments using r-tree

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
- Max test: explore best case performance (single file)

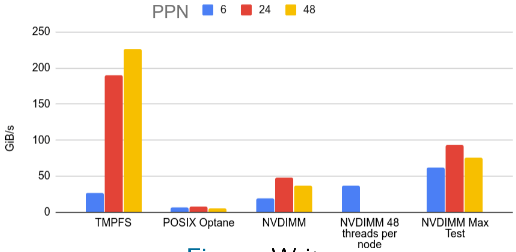


Figure: Write

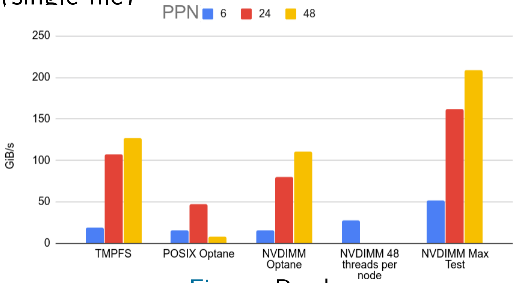


Figure: Read

Outline

Recent Improvements of ESDM

- 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 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

Summary

ESDM: Performance-portable I/O utilizing heterogeneous storage

1. The data model is mostly backwards compatible to NetCDF
2. NetCDF/Python workflows supported
3. Working toward workflow and active storage support
   - Exploiting **node-local storage** better
4. Next activities:
   - Comparison of flexible (ESDM) vs. fixed chunking (NetCDF)
   - Data re-mapping on read (transform-on-read) to optimize data access

Various other IO-related activities in ESiWACE

...

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

The Blocking I/O Path: Write

- Note: Processes write path is independent from any global state

1. Scheduler identifies how to partition the data into fragments and assigns backends
   - A maximum fragment size is defined by each backend
   - May also use a performance model to partition data
   - (We aim to utilize workflow information for the partitioning)
2. Append the fragment to the local dataset (mark as dirty)
3. A backend-specific thread pool processes the fragments
   - The backend is called with the fragment
   - May use direct I/O or reorganize the data in-memory
4. Wait until all fragments are processed

Collective operation

5. Upon close/sync, the MPI interface synchronizes the fragment knowledge
6. A single process updates the JSON metadata for the dataset/container

The Blocking I/O Path: Read

Preliminaries – Collective open/ref. operation of a dataset/container

1. Upon open, the fragment information is read by one process
2. Broadcast fragment information to all processes

3. Identify the overlap of fragments with the data space requested
4. Make a schedule to read each cell once (there could be replicas)
5. A backend-specific thread pool processes the fragments
   - Backend loads the fragments requested (use direct I/O or copy data if needed)
6. Wait until all fragments are processed

The ESiWACE1/2 projects have received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No **675191** and No **823988**

esiwace
CENTRE OF EXCELLENCE IN SIMULATION OF WEATHER AND CLIMATE IN EUROPE

Disclaimer: This material reflects only the author's view and the EU-Commission is not responsible for any use that may be made of the information it contains