2020 Summer School on Effective HPC for Climate and Weather

Input/Output and Middleware

Luciana Pedro, Julian Kunkel

Department of Computer Science, University of Reading

18 June 2020

Outline

1 Introduction

2 Input/Output

3 Middleware

4 I/O Solutions

5 I/O Performance

6 NetCDF

7 Parallel I/O in NetCDF-4

Learning Objectives

- Discuss challenges for data-driven research (Section Introduction)
- Describe the role of middleware and file formats (Section Middleware)
- Identify typical I/O performance issues and their causes (Section I/O Performance)
- Apply performance models to assess and optimize the application I/O performance (Section I/O Performance)
- Design a data model for NetCDF/CF (Section NetCDF)
- Describe ongoing research activities in high-performance storage (Section Research Activities)

Outline

1 Introduction
- I/O Bottleneck
- Data-driven Research

2 Input/Output

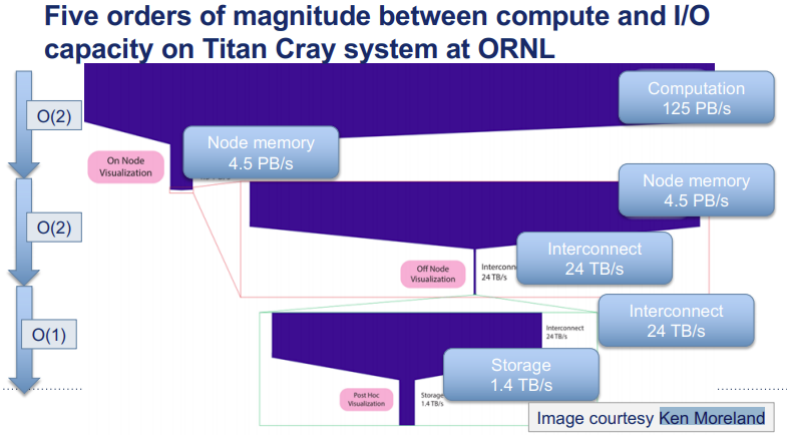
3 Middleware

4 I/O Solutions

5 I/O Performance

6 NetCDF

I/O Bottleneck – Example



LR: tut153s3

I/O Bottleneck – Historical Data

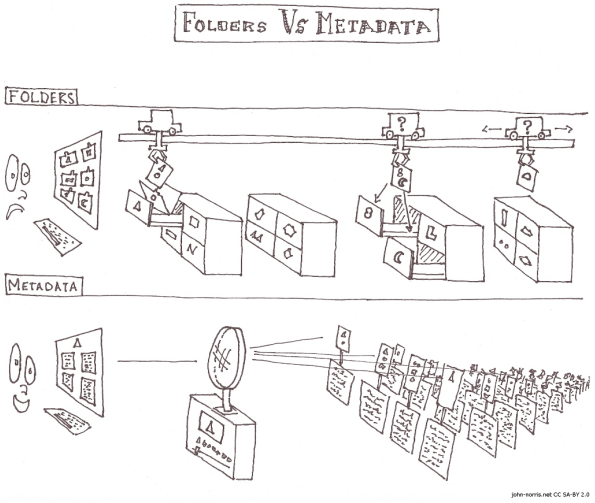
- Kunkel et al. [KKL14] analyze historical data from the German Climate Computing Center (DKRZ) and predict processor performance growth by 20x each generation (∼5 years), while storage throughput/capacity improves by just 6x.

Exascale Storage Systems – An Analytical Study of Expenses

	2004	2009	2015	2020	2025	Exascale (2020)
Performance	1.5 TF/s	150 TF/s	3 PF/s	60 PF/s	1.2 EF/s	1 EF/s
Nodes	24	264	2500	12,500	31,250	100k-1M
Node performance	62.5 GF/s	0.6 TF/s	1.2 TF/s	4.8 TF/s	38.4 TF/s	1-15 TF/s
System memory	1.5 TB	20 TB	170 TB	1.5 PB	12.8 PB	3.6-300 PB
Storage capacity	100 TB	5.6 PB	45 PB	270 PB	1.6 EB	0.15-18 EB
Storage throughput	5 GB/s	30 GB/s	400 GB/s	2.5 TB/s	15 TB/s	20-300 TB/s
Disk drives	4000	7200	8500	10000	12000	100k-1000k
Archive capacity	6 PB	53 PB	335 PB	1.3 EB	5.4 EB	7.2-600 EB
Archive throughput	1 GB/s	9.6 GB/s	21 GB/s	57 GB/s	128 GB/s	-
Power consumption	250 kW	1.6 MW	1.4 MW	1.4 MW	1.4 MW	20-70 MW
Investment	26 M€	30 M€	30 M€	30 M€	30 M€	$200 M[4]

Table 1. DKRZ System characteristics; future systems are a potential scenario

Folders vs Metadata



Data-driven Research

- **Data-driven Research** is the science of letting data tell us what we are looking for.
  - **Database Management** is the science of efficiently storing and retrieving data.
  - **Data Mining** is the science of discovering hidden correlations in data.
- In HPC, the concerns of **storage** and **computing** are traditionally separated and optimised independently from each other and the needs of the end-to-end user.
- Workflows composed of data, computing, and communication-intensive tasks should drive interfaces and hardware configurations to best support the programming models.
- Data-driven workflows may benefit from the explicit and simultaneous use of a locally heterogeneous set of computing and storage technologies.

Outline

1 Introduction

2 Input/Output
- Input/Output

3 Middleware

4 I/O Solutions

5 I/O Performance

6 NetCDF

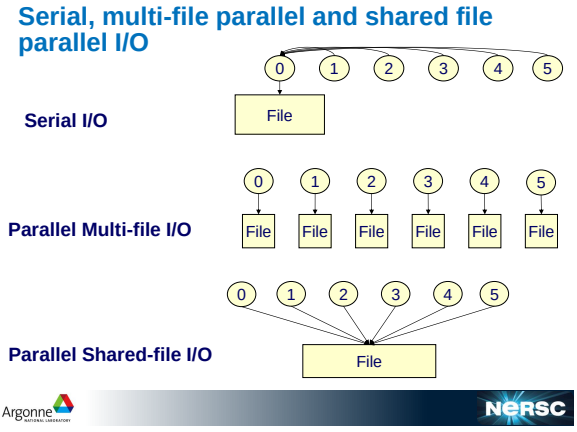
Input/Output

- Input/Output (I/O) is simply data migration.
  - Memory $\Leftrightarrow$ Disk
- I/O is a very expensive operation!
- How is I/O performed?
  - I/O Pattern
    - Number of processes and files.
    - Characteristics of file access.
- Where is I/O performed?
  - Characteristics of the computational system.
  - Characteristics of the file system.

I/O Performance

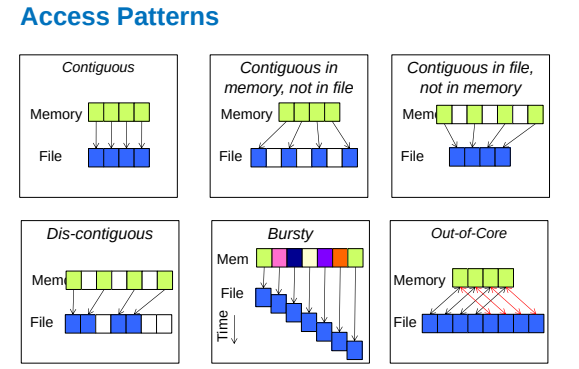
- There is no "One Size Fits All" solution to the I/O problem.
- Bottlenecks in performance can occur in many locations.
  - Application and/or file system.
- Many I/O patterns work well for some range of parameters.
- Going to extremes with an I/O pattern will typically lead to problems.
- **Golden Rule:** LR: Increase performance by decreasing the number of I/O operations (latency) and increasing size (bandwidth).

I/O Types



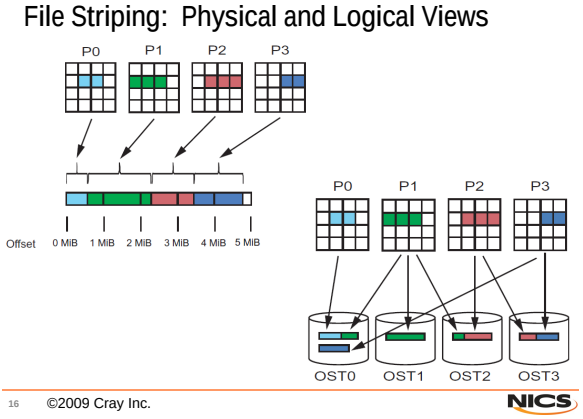
LR: tut145s3

I/O Access Patterns LR: (Double-check the last two)



LR: tut145s3

File Striping



I/O Problems

- Not enough I/O capacity on current HPC systems, and the trend is getting worse.
- If there is not enough I/O, you can not write data, so you can not analyze it.
  - Lost science!
- Energy consumption: it costs a lot of power to write data to disk.
- Opportunity for doing better science (analysis) when have access to full spatiotemporal resolution data.LR: What does that mean?

LR: tut153s3

Challenges in Application I/O LR: (== I/O Applications)

- LR: Leverage aggregate communication and I/O bandwidth of clients
  - LR: ... but not overwhelming a resource limited I/O system with uncoordinated accesses!
- Limit the number of files that must be managed.
- Avoid unnecessary post-processing. LR: (why?)
- Interact with storage through convenient abstractions.
  - Store in portable formats!
- **Golden Rule:** Parallel I/O software is available and they can address all of these problems, when used appropriately.

LR: tut145s3

Outline

1 Introduction

2 Input/Output

3 Middleware
- Introduction

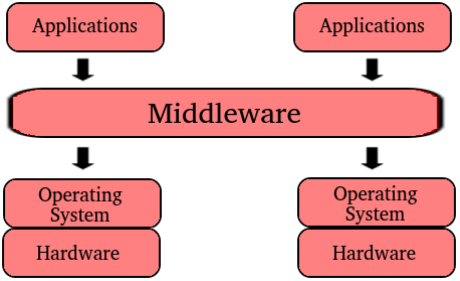
4 I/O Solutions

5 I/O Performance

6 NetCDF

Middleware

- Middleware is software occupying a middle position between application programs and operating systems.



Middleware

- Middleware is in the middle of the vertical stack, between the application programs and the operating system.
- Viewed horizontally rather than vertically, middleware is also in the middle of interactions between different application programs (possibly even running on different computer systems), because it provides mechanisms to support controlled interaction through coordination, persistent storage, LR: naming?, and communication.
- Middleware provide a more sophisticated form of persistent storage than the standard supported by most operating systems.LR: Is this true for all types of middleware?

Middleware – Examples

- Common middleware examples include relational database systems, application server middleware, message-oriented middleware, web middleware, and transaction-processing monitors.
  - Relational Database Management Systems (RDBMS): Oracle Database[1] and PostgreSQL[2].
  - Messaging Systems: IBM MQ[3], Jakarta Messaging API[4], and Java RMI (Remote Method Invocation)[5].
  - LR: Input/Output Systems: ESDM?

[1] https://www.oracle.com/database/technologies/
[2] https://www.postgresql.org/
[3] https://www.ibm.com/uk-en/products/mq
[4] https://projects.eclipse.org/projects/ee4j.jms
[5] https://www.oracle.com/java/technologies/javase/remote-method-invocation-home.html

Operating Systems and Middleware

- Operating systems and middleware have much in common.
- Both are software used to support other software, such as the application programs you run.
- Both provide a similar range of services centered around *controlled interaction*.
- **Controlled Interaction** is the interaction between concurrent computations.
  - On the same system – As between your email program and your word processor.
  - Across time – As between your word processor before your trip and your word processor after your trip.
  - Across space – As between your email program and your service provider's email server.

Operating Systems and Middleware

- Like an operating system, middleware may enforce rules designed to keep the computations from interfering with one another. An example is the rule that only one computation may modify a shared data structure at a time.
- Like an operating system, middleware may bring computations at different times into contact through persistent storage and may support interaction between computations on different computers by providing network communication services.
- However, operating systems and middleware are not the same.
- They rely upon different underlying providers of lower-level services.

Operating Systems and Middleware

- An operating system provides the services in its API by making use of the features supported by the hardware.
  - For example, it might provide API services of reading and writing LR: named LR: Named Binary Tag (NBT)?, variable-length files by making use of a disk drive's ability to read and write LR: numbered LR: ???, fixed-length blocks of data.
- Middleware, on the other hand, provides the services in its API by making use of the features supported by an underlying operating system.
  - For example, the middleware might provide API services for updating relational database tables by making use of an operating system's ability to read and write files that contain the database.
  - LR: ESDM!

Describe the role of middleware and file formats

- File formats
  -
  -
-
  -

Outline

1 Introduction

2 Input/Output

3 Middleware

4 I/O Solutions
- I/O Solutions
- Hardware Level
- Middleware Level
- Application Level

5 I/O Performance

I/O Solutions

- As we are moving towards exascale, the gap between computing power and I/O bandwidth will widen and researchers are looking for solutions to tackle this problem.
- There are essentially three lines of research:
  - at hardware level,
  - at middleware level,
  - and at application level.

Hardware Level

- Non-volatile memory (NVM)
  - Non-volatile memory (NVM) is a type of computer memory that can retrieve stored information even after having been power cycled.
  - The capabilities of NVM (i.e., capacity, bandwidth, energy consumption) are somewhere in-between main memory and persistent storage, thus it is often used as a "caching" solution between these two layers.
  - Examples of non-volatile memory include flash memory, read-only memory (ROM), ferroelectric RAM, most types of magnetic computer storage devices (e.g. hard disk drives, floppy disks, and magnetic tape), optical discs, and early computer storage methods such as paper tape and punched cards.

Hardware Level

- Burst buffer (BB)
  - Burst buffer (BB) is a fast and intermediate storage layer positioned between the front-end computing processes and the back-end storage systems.
  - HPC applications often show bursty I/O behavior (i.e., all processes read/write at the same time) and burst buffers help to absorb these workloads.
  - Burst buffer is built from arrays of high-performance storage devices, such as NVRAM and SSD.

NVRAM

SSD
(on top of
a hard drive)

Hardware Level LR: (Keep it?)

- Multi-layer Storage Hierarchy (Examples)
  - Attached SSDs to compute nodes to aggregate many small I/O requests into few larger ones and/or to compute nodes to speed-up MPI-IO.
  - Multi-layer storage hierarchy with NVM, SSDs, and different types of hard disks.
  - Fast Forward Storage and IO (FFSIO), SAGE, Distributed Application Object Store (DAOS), Post-Petascale File System (PPFS), Scalable Object-Centric Metadata Management (SoMeta), Extensible Metadata Provider for Extreme-Scale Scientific Simulations (EMPRESS), Týr
  - ECP [16], Fast-Forward [44], ADIOS [72], HDF VOL [29], ESiWACE [20], NEXTGenIO [69] and SAGE [78]

Middleware Level

- Solutions in I/O middleware.
  - E.g., file systems, I/O interfaces.LR: Is a file systems an I/O middleware?
- **Damaris:** Software framework that overlaps computation and I/O operations by dedicating a single core to I/O tasks.
- **ADIOS:** I/O abstraction framework for HPC applications that enables switching between different I/O transport methods with little modification to application code and enabling integration of new I/O solutions.
- **DeltaFS:** File systems that improves the scalability of file systems by letting compute nodes manage metadata instead of a centralized server.

Middleware Level

- ESDM

Application Level

- In-situ analysis
  - In biology and biomedical engineering, in situ means to examine the phenomenon exactly in place where it occurs (i.e., without moving it to some special medium).
  - Rather than applications writing their raw output to storage to later be read again for post-processing (e.g., visualization, filtering, statistics), in-situ processing removes this overhead by performing the analysis directly on the same machines as where the applications run.
  - ParaView, Dax, and Damaris/Viz are tools for large-scale in-situ visualization.LR: Use that style in middleware?

DiscussionLR: Keep just this slide?

- No one-size-fits-all solution to the storage problem and programmers must take I/O into careful consideration when developing applications.
- Mismatch between the massive computational performance of processors and relatively limited I/O bandwidth of storage systems.
- Three methods to alleviate this problem: new hardware technology, new I/O middleware, and application-specific solutions.
- Hardware technology shows promising solutions, but different systems might employ different solutions, reducing the portability and increasing the complexity.
- Middleware can alleviate some of this complexity with solutions LR: such as ESDM.
- In-situ analysis is an example of how application-specific solutions can be used to improve I/O throughput and thus application performance.

Outline

1 Introduction

2 Input/Output

3 Middleware

4 I/O Solutions

5 I/O Performance
- Introduction
- Typical Performance Factors
- I/O Performance Factors

I/O Performance

- There are several aspects involved in delivering high I/O performance to parallel applications, from hardware characteristics to methods that manipulate workloads to improve achievable performance.
- Running the same application with different I/O configurations gives the possibility to tune the I/O system according to the application access pattern.
- One way to predict application performance in HPC systems with different I/O configurations is by using modeling and simulation techniques.

I/O Stack

- I/O Hardware: Characteristics of Storage Devices affect performance, reliability, and system design.
- Parallel File System:
  - Manage storage hardware.
  - In the I/O software stack, focus on concurrent, independent access.
  - Publish an interface that middleware can use effectively.
- I/O Middleware:
  - Match the programming model (e.g. MPI).
  - Facilitate concurrent access by groups of processes.
  - Expose a generic interface.
  - Efficiently map middleware operations into operations in the Parallel File System.
- High Level Libraries
  - Match storage abstraction to domain.
  - Provide self-describing, structured files.
  - Map to middleware interface.
  - Implement further optimizations.
- Application

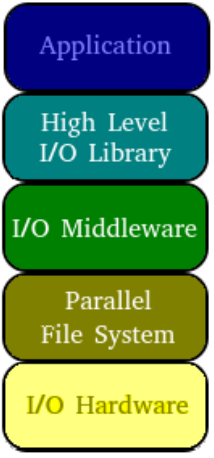
I/O Stack

I/O Middleware

- Match the programming model (e.g. MPI).
- Facilitate concurrent access by groups of processes.
- Expose a generic interface.
- Efficiently map middleware operations into operations in the Parallel File System.

Parallel File System

- Manage storage hardware.
- In the I/O software stack, focus on concurrent, independent access.
- Publish an interface that middleware can use effectively.



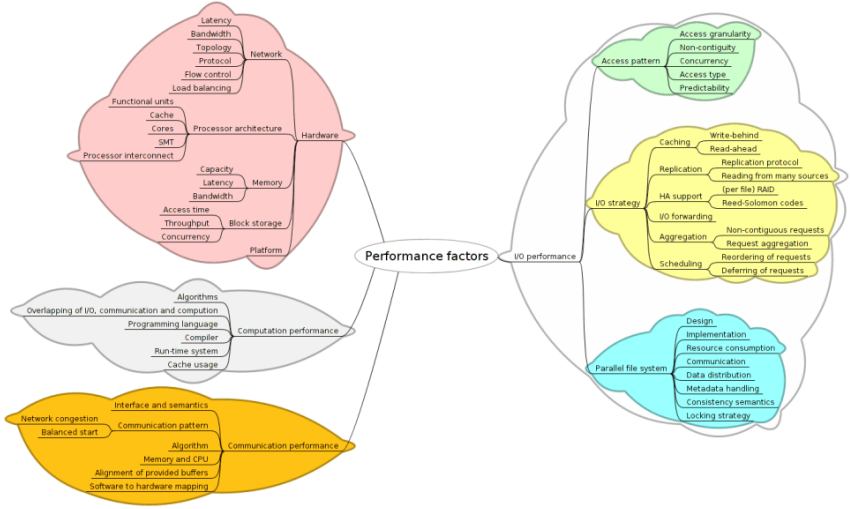
High Level I/O Library

- Match storage abstraction to domain.
- Provide self-describing, structured files.
- Map to middleware interface.
- Implement further optimizations.

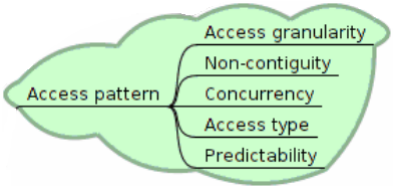
Application

- LR: Climate and Weather (choose some)

Typical Performance Factors



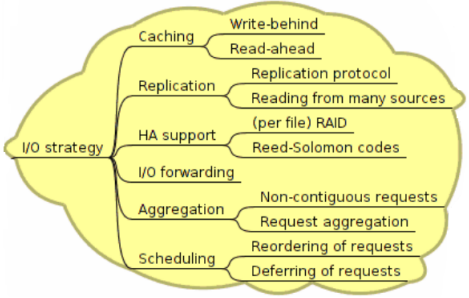
I/O Performance Factor – Access Patterns



- The access pattern describes how spatial access is performed over time.
- With an access pattern, the I/O of a single client process can be described, but also the actual observable patterns on the I/O servers, or on a single block device.
- The pattern on the I/O servers is caused by all clients and defines the performance of the I/O subsystems.

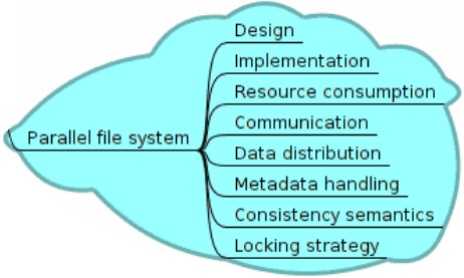
I/O Performance Factor – I/O Strategy

- In general, the mechanisms introduced here are orthogonal to the hardware and the architecture of the parallel file system.
- On the client-side, for instance, requests could already be tuned to improve the access pattern which will be observed on the servers.
- Similar to optimizations in communication, these strategies could be applied on any layer involved in I/O.



I/O Performance Factor – Parallel File System

- Performance of a parallel file system highly depends on its design as it provides the frame for the deployed optimization strategies.
- Several aspects like consistency semantics also apply to higher level interfaces like domain specific I/O libraries.



I/O Performance Tuning "Rules of Thumb"

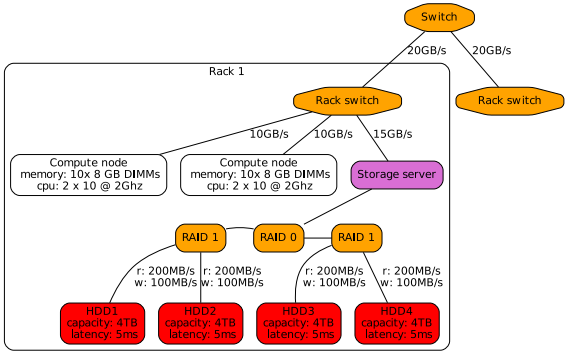
- Use collectives when possible
- Use high-level libraries (e.g. HDF5 or PNetCDF) when possible
- A few large I/O operations are better than many small I/O operations
- Avoid unnecessary metadata operations, especially `stat()`
- Avoid writing to shared files with POSIX
- Avoid leaving gaps/holes in files to be written later
- Use tools like **Darshan** to check assumptions about behavior

LR: tut145s3

I/O Performance Model

Example: Performance

Hierarchical model including different levels of detail.



I/O Performance Model

Example Scenarios for Mistral at DKRZ

	Mistral	Scale-down PFS, spent leftovers on compute		Switch to Object Storage, leftovers spent on compute	
Characteristics	Value	Factor	New value	Factor	New Value
Performance	3.1 PF/s	1.17	3.6 PF/s	1.19	3.7 PF/s
Nodes	2882	1.17	3370	1.19	3430
Node performance	1.0 TF/s				
System memory	200 TB	1.17	234 TB	1.19	238 TB
Network links	3100	1.12	3450	1.15	3565
Storage capacity	52 PB	0.5	26 PB	0.9	47 PB
Storage throughput	700 GB/s	0.5	350 GB/s	0.375	262 GB/s
Storage servers	130	0.5	65	0.75	98
Disk drives	10600	0.5	5300	0.74	7800
Archive capacity	500 PB				
Archive throughput	18 GB/s				
Compute costs	15.75 M EUR	1.17	19.53 M EUR	1.24	19.53 M EUR
Network costs	5.25 M EUR	1.10	6.04 M EUR	0.98	5.15 M EUR
Storage costs	7.5 M EUR	0.5	3.75 M EUR	0.5	3.75 M EUR
Archive costs	5 M EUR				
Building costs	5 M EUR				
Investment	38.5 M EUR		38.41 EUR		38.43 M EUR
Compute power	1100 kW	1.19	1290 kW	1.10	1309 kW
Network power	50 kW				
Storage power	250 kW	0.5	125 kW	0.75	188 kW
Archive power	25 kW				
Power consumption	1.20 MW		1.49 MW		1.57 MW

Case-study comparing Mistral as installed to a deployment with a reduced disk system and a deployment using object storage instead of a file system.

Jakob Luettgau, Julian Kunkel (DKRZ) HPC-IODC 2018 11 / 15

Assess and Optimize the Application I/O Performance

- Develop general considerations about what influences the I/O performance
  - What?
- Analyze access pattern and define how it defines the performance of the I/O sub-systems
  - How?
- Apply I/O strategies to improve the access pattern
  - Which?
- Identify options for the deployed optimization strategies in a specific parallel file system
  - Which?

LR: skill-tree, Julian needs to address it.

Outline

1 Introduction

2 Input/Output

3 Middleware

4 I/O Solutions

5 I/O Performance

6 NetCDF
- Introduction
- Common Data form Language (CDL)
- NetCDF Data Model

NetCDF

- In a simple view, NetCDF is:
  - A data model.
  - A file format.
  - A set of APIs and libraries for various programming languages.
- Together, the data model, file format, and APIs support the creation, access, and sharing of scientific data.
- NetCDF allows the user to describe multidimensional data and include metadata which further characterizes the data.
- NetCDF APIs are available for most programming languages used in geosciences.

Common Data form Language (CDL)

- The notation used to describe a NetCDF object is called CDL (network Common Data form Language), which provides a convenient way of describing NetCDF datasets.

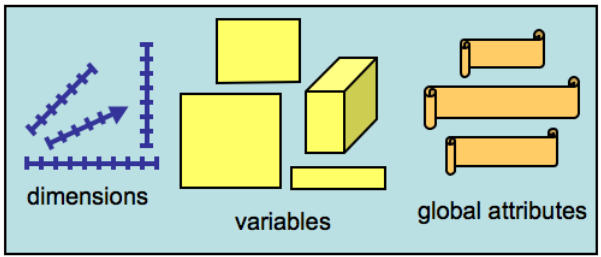
```
netcdf short {
dimensions:
  latitude = 3 ;
  longitude = 2 ;
variables:
  float sfc_temp(latitude, longitude) ;
    sfc_temp:units = "celsius" ;
data:

 sfc_temp =
  10, 10.1,
  10.2, 10.3,
  10.4, 10.5 ;
}
```

- The NetCDF system includes utilities for producing human-oriented CDL text files from binary NetCDF datasets and vice-versa.

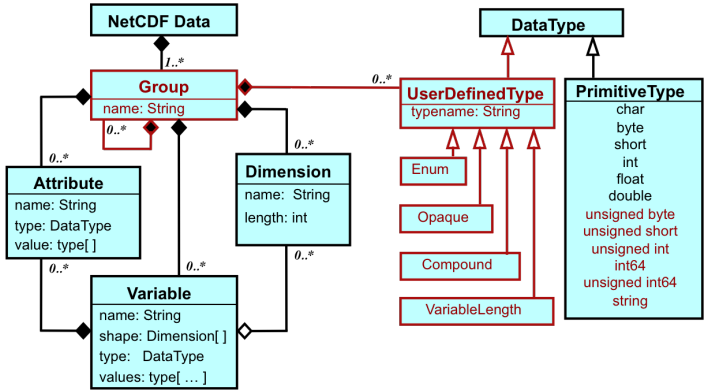
The Classic NetCDF Model

- A NetCDF file (dataset) has a path name and possibly some dimensions, variables, global (file-level) attributes, and data values associated with the variables.



NetCDF Data Models

- Classic: Simplest model – Dimensions, variables, attributes
- Enhanced: More powerful model – Adds groups, types, nesting



The NetCDF-4 Enhanced Data Model

- The NetCDF-4 Enhanced Data Model, which is known as the "Common Data Model", is part of an effort of Unidata to find a common engineering language for the development of scientific data solutions.
- The model contains the variables, dimensions, and attributes of the classic data model, but adds:
  - Groups – A way of hierarchically organizing data, similar to directories in a Unix file system.
  - User-defined types – The user can now define compound types (like C structures), enumeration types, variable length arrays, and opaque types.

The NetCDF-4 Enhanced Data Model

- A file has a top-level unnamed group.
- Each group may contain one or more named subgroups, user-defined types, variables, dimensions, and attributes.
- Variables also have attributes.
- Variables may share dimensions, indicating a common grid.
- One or more dimensions may be of unlimited length.

Experience-based "Best Practices" for Writing NetCDF Files

- Conventions
  - Developers should be familiar with and use existing NetCDF conventions.
- Coordinate Systems
  - Spatial and temporal location of data are supported by use of coordinate systems.
- Variable Grouping
  - How you group data into variables can determine whether common analysis and visualization software can effectively use the data.
- Variable Attributes
  - Conventional variable attributes supply necessary metadata.

Experience-based "Best Practices" for Writing NetCDF Files

- Strings and Character Variables
  - Use character data properly for representing text strings.
- Calendar Date and Time
  - Represent calendar dates and times with standards and conventions.
- Packed Data Values
  - Conventions for packing numeric data to save space have some subtleties.
- Missing Data Values
  - To indicate that data values are missing, invalid, or not written, special values are conventionally used.

Climate and Forecast (CF) Conventions

- The Climate and Forecast (CF) conventions are metadata conventions for earth science data, intended to promote the processing and sharing of files created with the NetCDF API.
- The purpose of the CF conventions is to require conforming datasets to contain sufficient metadata that they are self-describing:
  - Each variable in the file has an associated description of what it represents, including physical units if appropriate.
  - Each value can be located in space (relative to earth-based coordinates) and time.
- The CF conventions enable users of data from different sources to decide which data are comparable and allows building applications with powerful extraction, regridding, and display capabilities.

Outline

1 Introduction

2 Input/Output

3 Middleware

4 I/O Solutions

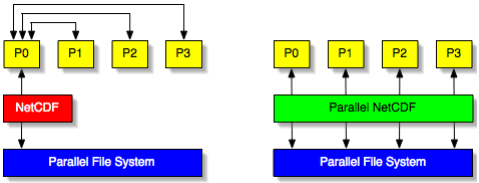
5 I/O Performance

6 NetCDF

7 Parallel I/O in NetCDF-4

Parallel I/O

- Parallel I/O allows each processor in a multi-processor system to read and write data from multiple processes to a common file independently.



- Data-intensive scientific applications use parallel I/O software to access files.
- In HPC, increasing demands in the I/O system can cause bottlenecks. Parallel I/O plays a fundamental role to balance the fast increase in computational power and the progress of processor architectures.
- Used properly, parallel I/O allows users to overcome I/O bottlenecks in HPC environments.

Parallel I/O in NetCDF-4

- NetCDF-4 provides parallel file access to both classic and NetCDF-4/HDF5 files.
- The parallel I/O to NetCDF-4 files is achieved through the HDF5 library while the parallel I/O to classic files is through PNetCDF.
- NetCDF-4 exposes the parallel I/O features of HDF5.
  - HDF5 provides easy-to-use parallel I/O feature.
- Parallel NetCDF uses MPI I/O to perform parallel I/O. It is a complete rewrite of the core C library using MPI I/O instead of POSIX.

Using Parallel I/O in NetCDF-4

- Special `nc_create_par` and `nc_open_par` functions are used to create/open a NetCDF file.
- The files they open are normal NetCDF-4/HDF5 files, but these functions also take MPI parameters.
- The parallel access associated with these functions is not a characteristic of the data file, but the way it was opened. The data file is the same, but using the parallel `open/create` function allows parallel I/O to take place.

```
EXTERNL int
nc_create_par(const char *path, int cmode, MPI_Comm comm,
              MPI_Info info, int *ncidp);

EXTERNL int
nc_open_par(const char *path, int mode, MPI_Comm comm,
            MPI_Info info, int *ncidp);
```

Collective and Independent Operations with Parallel I/O

- In MPI programs, I/O may be collective or independent.
  - Collective: It must be done by all processes at the same time
  - Independent: It can be done by any process at any time.
- All NetCDF metadata writing operations are collective. That is, all creation of groups, types, variables, dimensions, or attributes.
- Data reads and writes (ex. calls to `nc_put_vara_int` and `nc_get_vara_int`) may be independent (the default) or collective. To make writes to a variable collective, call the `nc_var_par_access` function.

```
/* Use these with nc_var_par_access(). */
#define NC_INDEPENDENT 0
#define NC_COLLECTIVE 1

EXTERNL int
nc_var_par_access(int ncid, int varid, int par_access);
```

Collective and Independent Operations with Parallel I/O

- You must build NetCDF-4 properly to take advantage of parallel features.
  - HDF5 must be built with `--enable-parallel`.
  - Typically the CC environment variable is set to mpicc, and the FC to mpifc (or some local variant.) You must build HDF5 and NetCDF-4 with the same compiler and compiler options.
  - The NetCDF configure script will detect the parallel capability of HDF5 and build the NetCDF-4 parallel I/O features automatically. No configure options to the netcdf configure are required.
  - For parallel builds you must include `"netcdf_par.h"` before (or instead of) `netcdf.h`.
  - Parallel tests are run by shell scripts. For testing to work on your platform the `mpiexec` utility must be supported.
  - Parallel I/O is tested in `nc_test4/tst_parallel.c`, `nc_test4/tst_parallel2.c`, and `nc_test4/tst_parallel3.c`, and other programs, if the `--enable-parallel-tests`

Parallel I/O Example – Part I

```
/* Initialize MPI. */
    MPI_Init(&argc,&argv);
    MPI_Comm_size(MPI_COMM_WORLD, &mpi_size);
    MPI_Comm_rank(MPI_COMM_WORLD, &mpi_rank);
    MPI_Get_processor_name(mpi_name, &mpi_namelen);

    if (mpi_rank == 1)
       printf("\n*** tst_parallel testing very basic parallel access.\n");

    /* Create a parallel netcdf-4 file. */
    if ((res = nc_create_par(FILE, NC_NETCDF4|NC_MPIIO, comm,
     info, &ncid))) ERR;

    /* Create two dimensions. */
    if ((res = nc_def_dim(ncid, "d1", DIMSIZE, dimids))) ERR;
    if ((res = nc_def_dim(ncid, "d2", DIMSIZE, &dimids[1]))) ERR;

    /* Create one var. */
    if ((res = nc_def_var(ncid, "v1", NC_INT, NDIMS, dimids, &v1id))) ERR;

    if ((res = nc_enddef(ncid))) ERR;
```

Parallel I/O Example – Part II

```
/* Set up slab for this process. */
start[0] = mpi_rank * DIMSIZE/mpi_size;
start[1] = 0;
count[0] = DIMSIZE/mpi_size;
count[1] = DIMSIZE;

/* Create phoney data. We're going to write a 24x24 array of ints,
   in 4 sets of 144. */
for (i=mpi_rank*QTR_DATA; i < (mpi_rank+1)*QTR_DATA; i++)
   data[i] = mpi_rank;

/*if ((res = nc_var_par_access(ncid, v1id, NC_COLLECTIVE)))
  ERR;*/
if ((res = nc_var_par_access(ncid, v1id, NC_INDEPENDENT))) ERR;

/* Write slabs of phoney data. */
if ((res = nc_put_vara_int(ncid, v1id, start, count,
   &data[mpi_rank*QTR_DATA]))) ERR;

/* Close the netcdf file. */
if ((res = nc_close(ncid))) ERR;

/* Shut down MPI. */
MPI_Finalize();

return 0;
```

Outline

1 Introduction

2 Input/Output

3 Middleware

4 I/O Solutions

5 I/O Performance

6 NetCDF

7 Parallel I/O in NetCDF-4

Describe ongoing research activities in high-performance storage

- TODO
-
-

Previous Learning Objectives

- Describe the general layers involved in I/O on a supercomputer
- Analyse the implications of parallel I/O on application efficiency
- Identify typical I/O performance issues and their causes
- Design a data model for NetCDF/CF
- Read, analyse, and write NetCDF files in a metadata-aware manner
- Visualise and regrid field constructs within NetCDF

Bibliography I

Sihem Amer-Yahia, Vincent Leroy, Alexandre Termier, Martin Kirchgessner, and Behrooz Omidvar-Tehrani.
Interactive Data-Driven Research: the place where databases and data mining research meet.
Research Report RR-LIG-049, LIG, 2015.
Les rapports de recherche du LIG - ISSN : 2105-0422.

Lonnie Crosby.
Parallel I/O Techniques and Performance Optimization, 2012 (Accessed in August 2020).

Stijn Heldens, Pieter Hijma, Ben Van Werkhoven, Jason Maassen, Adam S. Z. Belloum, and Rob V. Van Nieuwpoort.
The landscape of exascale research: A data-driven literature analysis.
ACM Comput. Surv., 53(2), March 2020.

Julian Kunkel, Michael Kuhn, and Thomas Ludwig.
Exascale Storage Systems – An Analytical Study of Expenses.
Supercomputing Frontiers and Innovations, pages 116–134, 06 2014.

Julian Kunkel and Luciana Pedro.
Potential of I/O Aware Workflows in Climate and Weather.
Supercomputing Frontiers and Innovations, 7(2), 2020.

Julian Kunkel.
Simulation of Parallel Programs on Application and System Level.
Phd thesis, Universität Hamburg, 07 2013.

UCAR/Unidata.
Network Common Data Form (NetCDF), Accessed in August 2020.

Bibliography II

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**

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